Fachhochschule Konstanz
Fachbereich Informatik

Masterthesis

zur Erlangung des akademischen Grades

Master of Computer Science

an der

Fachhochschule Konstanz

Hochschule für Technik, Wirtschaft und Gestaltung

Fachbereich

Informatik / Wirtschaftsinformatik

Thema:

„Integration von Legacy Systemen in das Internet durch die Exposition von Transaktionen als Web Services.“

Verfasser : Servet Catal, Dernburgweg 21 in 64289 Darmstadt
Firma : Lufthansa Systems, am Weiher 24, 65451 Kelsterbach
Betreuer : Professor Dr. rer. pol. Paul Wenzel & Dipl. Phys. Robert Jürgens
Eingereicht : Konstanz, 18. Oktober 2002

VORWORT

Bei Recherchen für ein geeignetes Diplomarbeitsthema wurde ich auf der Homepage von Lufthansa fündig. Die von Herrn Dipl. Phys. Robert Jürgens gestellte Aufgabenstellung entsprach ganz meinen Vorstellungen.

Bei der Ausarbeitung der vorliegenden Diplomarbeit habe ich viele interessante und lehrreiche Einblicke in die Dynamik der IT-Welt gewinnen können, insbesondere die weitreichenden Möglichkeiten der Webservices haben mich für meinen beruflichen Werdegang stark geprägt.

DANKSAGUNG

Mein Dank gilt Herrn Prof. Dr. rer. pol. Paul Wenzel für seine Betreuung der Diplomarbeit, Herrn Korreferent Dipl. Phys. Robert Jürgens für seine konstruktiven Ratschläge und Bereitstellung von Arbeitsmitteln, meinem Cousin Herrn Utkan Durgel für seine redaktionelle Beihilfe und nicht zuletzt meiner Ehefrau und beiden Kindern, für ihre psychologische Unterstützung und ihre Geduld.

SERVET CATAL

DARMSTADT, IM OKTOBER 2002

INHALTSVERZEICHNIS

Abkürzungsverzeichnis

DB	Datenbank
DBMS	Datenbank-Management-Systeme
GL	Geschäft-Logik
IT	Information Technology
SAML	Security Assertion Markup Language
SL	Service-Logik
SOAP	Simple Object Access Protocol
u.a.	unter Anderem
UDDI	Universal Description, Discovery and Integration
u.U.	unter Umständen
W3C	Web-3-Concortium
WS	Webservice
WSDL	Web Services Description Language
WSFL	Web Services Flow Language
WSUI	Web Services User Interface
XACML	Extensible Access Control Markup Language
XAML	Transaction Authority Markup Language
XKMS	XML Key Management Specification
XLANG	Web Services for Business Process Design
z.B.	zum Beispiel
z.Z.	zur Zeit

1. EINLEITUNG

Innovation, Geschwindigkeit und Zusammenarbeit sind die wichtigsten Kriterien für die heutige wirtschaftliche Welt, wenn es darum geht, einerseits bestehende Märkte zu sichern und auszubauen, andererseits neue Märkte zu erschließen. Um in einem sich immer schneller ändernden Markt zu bestehen, brauchen Firmen Business-Applikationen, die flexibel und kooperativ auf interne wie externe Bedürfnisse und Veränderungen reagieren. Jedoch erfüllen die heute gebräuchlichen Informationssysteme mit ihren starren Strukturen die Wünsche nicht in diesem Maße.

Diese Alt-Systeme, die oft ein Vermächtnis (Legacy) der letzten Dekaden darstellen, steuern die Arbeitsprozesse- und Datenressourcen, die sich all in den Jahren entwickelt haben. Sie müssen ständig den Bedürfnissen von Produktion und Dienstleistung angepasst werden. Damit sie nun die neuen Anforderungen von heute erfüllen können, müssen sie in moderne Softwaresysteme integriert werden, auch wenn man sie aus wirtschaftlichen und technischen Gründen weiterhin betreiben muss.

Dabei liegt die Fokussierung vieler IT- Verantwortlichen auf der Mainframe-Integration von unternehmensintern etablierten, gleichzeitig geschäftskritischen Anwendungen und Techniken, die unverzichtbar behütet und gepflegt werden, jedoch auch den Anschluss an innovative Internet-Technologien ohne Mehrinvestitionen finden müssen, um potenziellen Nutzen aus unternehmensübergreifenden Transaktionen zu erlangen.

Das Internet spielt dabei mit ihren weitreichenden Möglichkeiten eine Schlüsselrolle. Für die Integration unterschiedlicher Systeme im Web müssen gemeinsame Standards geschaffen werden, damit die Plattform- und Sprachabhängigkeit von konkurrierenden Produkten unterbunden werden kann.

Hierfür bietet man nun die Web Services verheißungsvoll an. Sie werden unlängst als Allheilmittel gegen alle Integrationsprobleme angepriesen. Ein kritischer Blick hinter die Kulissen offenbart das wahre Potential: Sie erringt gegenüber den bisherigen Integrations-bemühungen namhafter Hersteller leicht Vorteile, auch die Softwareindustrie-Titanen unterstützen und entwickeln nunmehr massiv die Webservice-Applikationen, nachdem sie die Grundlagen dessen gemeinsam geschaffen haben. So wird sie nicht nur für die nächste Zeit eine große Rolle spielen, auch wenn heute einige Anforderungen (z.B. Sicherheit) nicht gänzlich erfüllt sind.

1.1 AUFGABENSTELLUNG

Thema: ***„Integration von Legacy Systemen in das Internet durch die Exposition von Transaktionen als Web Services.“***

Konzept: Die Nutzung von Transaktionen durch das Internet mittels eines Browsers ist durch den Hersteller (UNISYS) mit der Einführung eines Webservers auf dem Host ermöglicht worden (WebTS). Die Anpassung der Legacy Anwendung erlaubt den Schedule einer Transaktion durch einen nach dem HTTP-GET Protokoll formatierten URL.

Im Rahmen dieser Diplomarbeit soll nun eine ASP.Net basierende Anwendung erstellt werden, durch die neben der Autorisierung eine Auswahl von Transaktionen mit Hilfe eines Webservices über eine Standalone-Anwendung als auch über aktive Webpages ausgeführt werden können.

Für die Entwicklung sollte das Microsoft .NET SDK bzw. Visual Studio .NET eingesetzt werden. Die Programmiersprache für den Client kann frei gewählt werden, während C# für die Beschreibung der Webservices und Proxyklassen empfohlen wird.

Folgende Schritte sind durchzuführen:

- Erstellen eines LOGON Webservice, durch die eine Autorisierung gegenüber dem Legacy Systeme ermöglicht wird.
- Erstellen eines AD (Agent Display) Webservice, durch die eine Erfolgskontrolle der Autorisierung ermöglicht wird.
- Erstellen eines TB (Terminal Information) Webservice, welcher ein Abfrageservice ist, der mehrere Transaktionsaufrufe zusammenfasst.
- Erstellen eines LOGOUT Webservice.

Die Skalierbarkeit dieser Architektur im Hinblick auf hohe Zugriffslast soll dargestellt und der Einsatz von Standard-Technologie zur Sicherung des Zugriffs wie Verschlüsselung und Zertifikate belegt werden. Ein Kostenvergleich mit bisherigen Technologien ist anzufügen.

In dieser Arbeit wurde eine typische Abfrage aus dem Flugbetrieb anhand einer Lufthansa-Passagierliste (NPL) praktisch abgebildet. Dabei fand innerhalb der Microsoft .NET SDK Plattform C# als Programmiersprache für den Client sowie für die Beschreibung der Webservices und der Proxyklassen Anwendung.

1.2 ZIELSETZUNG

Das Ziel dieser Arbeit besteht darin, aufzuzeigen, dass Businesslogik der Legacy-Mainframe-Systeme mittels vorhandener Webservice-Techniken im Internet exponiert werden können.

- Das primäre Ziel ist die Verarbeitung von Transaktionsdaten durch ASP.NET
- Das sekundäre Ziel ist die Darstellung der Daten im Browser.

Bevor mit der praktischen Durchführung begonnen werden kann, muss zuerst geklärt werden, was überhaupt Web-Services sind (im Folgendem mit WS abgekürzt) und was sie so attraktiv macht. Schließlich wollen und müssen alle renommierten Softwarehersteller rechtzeitig auf den fahrenden Zug dieser „revolutionären“ (oder doch evolutionär?) Technologie“ aufspringen, die gerne mit dem Anspruch verbunden wird, die Integrations-probleme des Internet- Zeitalters plattform- und sprachunabhängig zu lösen.

2. INTEGRATION VON LEGACY SYSTEMEN IN DAS INTERNET

2.1 KLASSISCHE HOST-ANWENDUNGEN

Die meisten Legacy-Systeme werden mit Host-Rechnern innerhalb eines Netzwerkes betrieben, deren Mainframe-Architektur auf der Streuung des Zugriffsports basiert.

Viele Terminals sind an einem Großrechner zentral angebunden. Die Datenverarbeitung erfordert hohe Rechenleistung, da alle möglichen Funktionen dem Mainframe aufgebürdet werden. Die Client-Terminals sind lediglich für eine dezentrale Ein- und Ausgabe zuständig.

Mainframe-Server haben noch ihre Berechtigung bei Aufgaben, wo enorme Daten in kürzerster Zeit mit geringem Verwaltungsaufwand ausfall- und zugriffssicher verarbeitet werden müssen.

Die weit verbreiteten Architekturen SNA von IBM, ClearPath von Unisys u.a., deren Protokolle man bei der Benutzung von Host Rechnern zwangsläufig nicht missen möchte, stoßen auf Probleme des netzwerkübergreifenden Datenaustausches auf heutigen Netzen. Diese setzen oft das Protokoll TCP / IP ein, das für universelle Kommunikation intelligenter Geräte entwickelt wurde, die Netzwerkarchitekturen jedoch die Kommunikation von dummen Terminals mit dem Großrechner und nicht die der intelligenten Rechner untereinander vorsahen.

2.2 CLIENT-SERVER-ARCHITEKTUREN

Die folgende Abbildung zeigt den Verlauf der Software-Entwicklung von Anwendungs-systemen, der sich in letzten Jahren vollzogen hat. Insbesondere seit der Einführung des Internets werden immer neue und standardisierte Konzepte zur Entwicklung von Client/Server-Anwendungen benötigt.

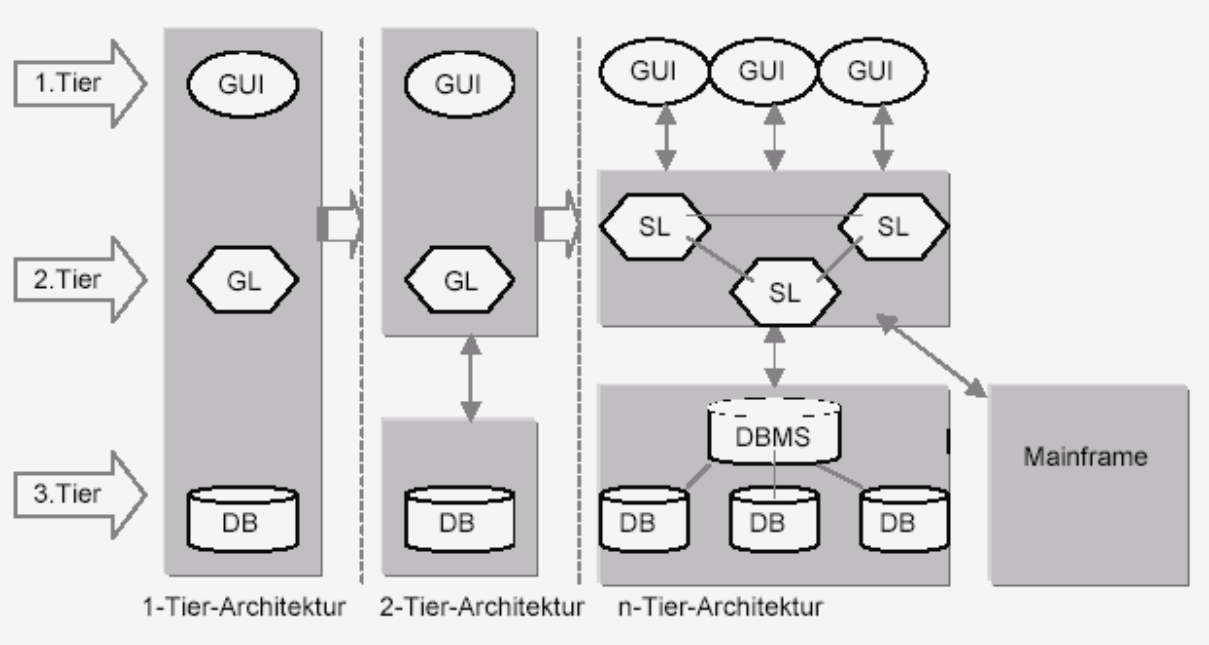


Abbildung 1 - Entwicklungsstufen von Tier-Architekturen

Monolithische Anwendungen (Ein-Tier-Architektur)

Diese traditionellen Enterprise-Anwendungen sind abgekapselte Programme, die auf einer monolithischen (eigenständigen) Struktur basieren. Sie erlauben lediglich eingeschränkte Zugriffe auf die Daten und Prozeduren zueinander. Sie sind in der Regel von der Wartung her relativ schwer zu handhaben, weil jede kleine Änderung ggf. eine Neu-Strukturierung und- Kompilieren dieser Programme erfordert.

Zwei-Tier-Architektur

Hierbei werden die Rechenkapazitäten des Servers auf die Client-Rechnern ausgelagert und so der Server entlastet. Dafür kommt auf dem Server eine Datenbank zum Einsatz, während auf dem Client alle restlichen Funktionalitäten wie die Logik und Darstellung abgelegt werden. Demzufolge müssen die Anwendungen auf allen Arbeitsplätzen installiert sein. Jeder Benutzer verfügt somit über die gleichen Funktionen und die gleiche Datenbank (große Netzbelastung!), wobei der Einsatz an verschiedenen Orten liegen kann. Viel Anwendungslogik im Client bedingt aber hohe Kosten für die nachfolgende Wartung.

Multi-Tier-Architektur

Aufbauend auf das 3-Tier-Architektur, wo die allgemeine Logikschicht auf den Server ausgelagert und meist nur noch die Präsentationsschicht auf dem Client mit seiner individuellen Logik belassen wird, sind nunmehr alle Komponenten, Anwendungslogik und Serverdienste verteilt. So kommt man mit weniger Speicher aus. Die Clients holen sich dabei die benötigten Programme vom Server. Die Konsequenz aus so einer Anwendungs-architektur ist neben geringeren Kosten auch eine wesentlich effektivere Softwareverteilung. Neue Programmversionen brauchen nicht mehr dezentral auf jedem einzelnen Arbeitsplatz installiert zu werden, der Anwender kann immer auf die neueste Version zugreifen, die vom Server bereitgestellt wird.

Die Aufgaben der Schichten im Detail:

1.Tier: Graphische Benutzeroberfläche (GUI)

Sie definiert Schnittstellen für Benutzerinteraktionen und zielt vor allem auf eine ansprech-bare, einfach zu bedienende graphische Benutzeroberfläche ab, die vom Benutzer entweder lokal oder im Netz bedient wird.

2.Tier: Geschäftslogik (Service-Logik)

Die Clients interagieren mit den auf dem Server befindlichen Geschäftsobjekten (Business Objects). Sie sind für die logischen Geschäftsfunktionen der Anwendung verantwortlich.

3.Tier: Datenbanken

Die Datenbank-Schicht besteht im wesentlichen aus DBMS-Produkte, die von den Geschäft-sobjekten der zweiten Schicht genutzt werden. Der Zugriff der Clients auf die Datenbanken kann nur über die Geschäftsobjekte laufen.

2.3 LEGACY-SYSTEME UND WEGE ZUR INTEGRATION

Die meisten Legacy Systeme sind in sich geschlossene Anwendungen, die eine Kommunikation mit anderen außenstehenden Systemen verhindern. Neue Geschäftsmodelle erfordern aber den Austausch von Informationen zwischen verschiedenen Anwendungen. Durch wachsende Kostendruck und die Forderung nach Investitionsschutz stellt sich nun die Frage, wie sich die bestehenden Systeme mit möglichst geringem Aufwand in fremde oder neue Systeme einbinden lassen.
Aus technischer Sicht bieten sich drei Möglichkeiten an:

- Neu-Entwicklung (Redevelopment)
- Migration (Reorganisation)
- Integration

Eine Neuentwicklung der Funktionen oder eine Reorganisation der Datenhaltung würde aufgrund der gewachsenen Komplexität dieser Systeme horrende Kosten verursachen und sehr viel Zeit in Anspruch nehmen. Es kann unter Umständen Jahre dauern, bis die „neuen alten“ Funktionen wieder den Grad von Zuverlässigkeit erreicht haben, den sie früher im Legacy-System hatten. Auch müssen Legacy-Systeme oft aus wirtschaftlichen und technischen Gründen weiter betrieben werden und unterliegen daher auch regelmäßigen Systemanpassungen, die bei einer Neurealisierung der Systeme immer wieder in den neu entwickelten Funktionen nachvollzogen werden müssten.

Selbst die erfolgreichsten Migrationsmodelle mutieren mit der Zeit zu neuen Legacy-Systemen, wenn sie zukünftige Entwicklungen nicht berücksichtigen können.

Aus diesen Gründen verlagert man die Bemühungen weg von der Migration hin zur Integration von Legacy- Systemen: Man versucht, bestehende Funktionen oder existierende Datenbestände ohne gravierende Anpassungen zu nutzen. Vgl. [Gruhn02]

Dabei werden dem Integrator viele Stolpersteine in den Weg gelegt, u.a.:

- Fehlende oder unzureichende Dokumentation von Datenquellen und Kommunikationsverbindungen.
- Viele unterschiedliche Systeme, Anwendungen und Quelldaten kommunizieren über nicht standardisierte Schnittstellen.
- Zeit- und kostenintensive Pflege und mangelhafte Erweiterungsfähigkeit der Systeme.
- Eine firmenübergreifende Integration z.B. bei Unternehmensfusionen wird ohne gemeinsame Standards erschwert.

Geschäftsanforderungen für eine erfolgreiche Integration:

- Die neuen Applikationen sollten für alle Anwender gleichermaßen einfacher zu bedienen sein als ihre bisherigen Geschäftsmethoden.
- Systemübergreifende Geschäftsprozesse müssen ermöglicht werden.
- Echtzeit-Zugriff auf geschäftliche Daten und Informationen muss für Kunden und Zulieferer erfüllt werden.
- Investitionsamortisierung (ROI) muss durch rasche Umsetzung und schnelle Markteinführung von neuen Applikationen erfolgen.

Technische Anforderungen an die Integration:

- Bestehende Systeme müssen ohne Unterbrechung des laufenden Betriebes sukzessive in die Applikation integriert werden.
- Alle signifikanten Daten im Unternehmen müssen vollständig integriert werden.
- Vorhandene Systeme dürfen wegen Stabilität keine Veränderungen erfahren.
- Die Integration muss durch Anwendung von modernen Standards zukunftssicher gestaltet werden.

Zugriff auf die Legacy-Daten

Der Zugriff auf die Host- Daten lässt sich durch 5 verschiedene Wege ermöglichen.

Direkt (Invasiv): Änderungen am Host oder Zugriff auf einer niedrigen Ebene

- Ermöglicht oder vereinfacht den direkten Zugriff auf die Mainframe-Datenbanken.
- Verwendet für den Zugriff auf die Mainframe-Daten ein Application Programming Interface (API).
- Unterzieht die Anwendung einem Re-Engineering, so dass die Daten und von den Benutzern geforderte Anwender-Interface bereitgestellt werden.

Indirekt (non invasiv): Zugriff außerhalb des Hosts

- Verwendet für den Zugriff das meistbenutzte Protokoll tn-3270 von IBM. Dieses Verfahren wird auch als "Screen-Scraping" bezeichnet.
- Kombination von Screen-Scraping mit modernen, objektorientierten und server-basierten Technologien. Dabei kommt ein mehrschichtiges Anwendungsmodell (Multi-Tier) zum Einsatz.

Direkter Zugriff auf Mainframe-Datenbanken

Generell liegen die Host-Daten in irgendeiner Art von Datenbank und sind oft äußerst kryptisch abgespeichert. Z.B. sind die Anwendungen vor vielen Jahren erstellt worden, als Speicherplatz noch teuer war. Die komprimierten Informationen sind daher meistens schwer nachvollziehbar. Um also direkt auf die Datenbank zugreifen zu können, müssen diese nicht immer gut dokumentierten Verschlüsselungs-Schemata erst einmal entziffert werden.

Außerdem muss der Entwickler die Geschäftslogik replizieren, welche die Daten für die laufende Geschäftstransaktion in ein brauchbares Format umwandelt. Für Aktualisierungen gilt natürlich der umgekehrte Weg. Die ursprüngliche Host-Software verwendet beispielsweise eine spezielle Methode, um zwei Kunden mit gleichem Namen zu verwalten. Wenn die Beziehungen von Daten schlecht dokumentiert sind, ist ihre Integrität und die Einhaltung der Regeln in Gefahr.

Dritte Voraussetzung ist das Verständnis des gesamten Applikation-Systems, das die für den Zugriff benötigte Datenbank umgibt. Wenn jedoch eine neue Applikation die Datenbank ohne Synchronisation der ursprünglichen Legacy-Updates direkt aktualisiert, treten enorme Probleme auf. Wer aber die Datenformate und Wechselbeziehungen zwischen den

Anwendungen gut kennt und die Geschäftslogik weitgehend einfach replizieren kann, ist mit dem direkten Datenbankzugriff gut bedient.

API-Zugriff auf Mainframe-Daten

Ein anderer Weg ist der Zugriff auf Host-Daten über ein API:

Das IBM "External Presentation Interface (EPI)" ist für die meisten CICS-Anwendungen erhältlich und erfordert Entwicklerkenntnisse des 3270-Datenstroms. Die weiteren IBM-Entwicklungen "External Call Interface (ECI)", das "Common Programming Interface for Communications (CPI-C)" und die MQ-Series mit Message-Queuing-API bieten auch Zugriff auf Host-Daten. Falls eine Applikation eine dieser Zugriffe anwendet, sollte u.U. auch weiterhin diesen Weg beschreiten. Wenn eine Anwendung einen API-Zugriff besitzt, ist das oft die beste Methode für das Auffinden der Daten. Die APIs können zudem integrierte Mechanismen für die Datenintegrität und die Synchronisation bieten, die gewährleisten, dass eine Nachricht wirklich nur einmal verschickt wird.

Re-Engineering

Mit ausreichend Zeit, Geld und Programmierern ist es natürlich kein Problem, die Host-Anwendung komplett neu zu schreiben und sie gleichzeitig mit einer modernen Benutzer-oberfläche auszustatten. Dies ist ein gangbarer Weg, wenn die Applikation komplett überarbeitet und mit wesentlichen Funktionalitätssteigerungen ausgestattet werden soll. Für das Re-Engineering ist das Neuschreiben der Geschäftslogik unumgänglich. Dies kann dann gewünscht sein, wenn das gegenwärtige Geschäftsmodell radikale Änderungen erfahren hat und das Unternehmen lieber einen völligen Neubeginn wagen möchte, als das Vorhandene wirksam zu verbessern. Wenn jedoch Kapital, Ressourcen und Erfahrungen limitiert sind, die Anwendung bereits teilweise aktualisiert wurde oder rasche Markteinführung notwendig ist, darf Re-Engineering nicht angewendet werden.

Non invasive Schritte zur Legacy-Erweiterung

Wenn das Host-System für den externen Zugriff nicht modifiziert oder neu konfiguriert werden darf, gleichgültig der physikalischen, verfahrenstechnischen oder politischen Gründe, ist in jedem Fall eine der folgenden nicht invasiven Strategien anzuwenden.

Screen-Scraping

Dabei übernimmt eine maßgeschneidert programmierte Anwendung die Kontrolle über eine Emulator-Session wie 3270, 5250 oder VT100/220. Sie sendet und empfängt Datenströme, die der Host als Kommunikation für jeden normalen Host-Bildschirm interpretiert. Mit Hilfe von IBM-Emulatoren wie "High Level Language Applications Programming Interface (HLLAPI)" können dann aus diesen Datenströmen die Informationen, die der Anwender bearbeiten möchte, herausgezogen werden. Ein Nutzen der Legacy-Funktionen ohne jegliche Änderungen des Systems ist somit möglich. Jedoch macht die kleinste Anpassung am Legacy-System den relativ geringen Entwicklungsaufwand dieses APIs zunichte, da daraus umso höheres Wartungsaufwand durch Anpassung am API resultiert. Ältere Screen-Scraper differieren stark in ihrer Leistungsfähigkeit, Skalierbarkeit und Robustheit. Im Zwei-Tier-Ansatz kommuniziert das Client-Programm, welches die neue Anwenderschnittstelle bereitstellt, direkt mit dem Legacy Backend. Heute arbeiten die Produkte mit weniger komplexen, aber dennoch programmierintensiven APIs, mit denen Entwickler API-Calls von ihren bevorzugten Entwicklungsumgebungen, wie Visual Basic, Java oder PowerBuilder, einsetzen können.

Unternehmen, die diese schwierigen und ermüdenden Programmierbürden auf sich genommen haben, sehen einige Vorteile dieser Methode:

- Die Host-Anwendungen müssen nicht modifiziert werden. Angesichts der Tatsache, dass keiner den Legacy-Code und die Datenstrukturen wirklich versteht, ein enormer Vorteil. Selbst wenn die System-Kenntnisse vorhanden sind, hinken die meisten IT-Abteilungen mit ihren Host-Applikationsprojekten hinterher, da sie die begrenzten Ressourcen nicht noch weiter aufteilen können.

- Die Datenströme der Screens machen den Zugriff leicht. Normalerweise bietet eine screen-basierte Anwendung die korrekte Geschäftslogik für den Datenzugriff. Die interne Datenbankstruktur des Hosts und die Applikationslogik müssen nicht bekannt sein, um sicherzustellen, dass die neue Anwendung alle erforderlichen Schritte ausführt.

Multi-Tier-Anwendungsmodellierung

Die Weiterentwicklung des "Screen-Scraping"-Konzeptes von der Zwei-Tier-Methode zur Drei-Tier-und schlussendlich zum Multi-Tier-Architektur verwendet ein Server als zusätzliche Ebene zwischen dem Client und den Legacy-Anwendungen, deren Daten isoliert von den Client-Applikationen bearbeitet und verwaltet werden können. So kann die Anwendungslogik und die Datenhaltung an verschiedene Orte und in unterschiedliche Rechensysteme verteilt werden; die Lokalisierung eines Geschäftsprozesses spielt in diesem vernetzten System eine untergeordnete Rolle.

Bei Anwendungsänderungen sind lediglich Modifikationen am Server-System notwendig, eine Neuprogrammierung und Verteilung der Clients entfällt. Viele Client-Typen können auf denselben server-basierten Code und die gleichen Host-Screen-Modelle zugreifen; z.B. können Mitarbeiter von einer nur intern verfügbaren CRM-Lösung die Legacy-Daten nutzen, während Kunden ähnliche, aber „filtrierte“ Informationen über das Web bekommen.

Die Multi-Tier-Umgebung ermöglicht somit den flexiblen, skalierbaren und einfach zu verwaltenden Anwenderzugriff nahezu in Echtzeit und eine zügige Marktplatzierung mit geringerem Budget. Sie unterstützt eine Vielzahl von Anwendungen, die u.a. mit Java, Windows-APIs, C-Familie und neuerdings über XML laufen. Vgl.[Jacada]

2.4 INTEGRATIONSTECHNOLOGIEN FÜR LEGACY-SYSTEME

In einem Legacy System entstanden im Laufe der Zeit unzählige Anwendungen, die teilweise nicht für eine Zusammenarbeit entworfen wurden und auch nur Teilaufgaben von Geschäfts-prozessen lösen. Diese heterogenen Anwendungen zu vereinheitlichen, um den aktuellen Geschäftsprozessen zu entsprechen, ist die Aufgabe von **EAI** (Enterprise Application Integration). EAI- Softwareprodukte integrieren die Anwendungen eines Unternehmens, indem sie Middleware und Kapselung benutzen.

Middleware umfasst alle Dienstprogramme außer Basis-und Anwendungssoftware; sie verbindet Anwendungsprogramme miteinander und führt gewisse gewöhnliche Dienstfunktionen für sie aus. Durch ein Wrapper kann man nun die innere Ausprägung dieser Programme hinter einer klar definierten Schnittstelle verbergen.

Legacy Anwendungen besitzen üblicherweise keine gängigen Schnittstellen- Middleware im Gegensatz zu Standardanwendungen mit wohldefiniertem API (z.B. SAP R/3 mit einer RPC basierten API), die leichten Zugriff durch Client-Software-Komponenten ermöglichen. Um eine Integration trotzdem zu ermöglichen, muss ihnen eine zusätzliche

„Softwareschicht“, ein **Wrapper** zugefügt werden, der eine Art Standardschnittstelle erzeugt und somit die Implementierung davon trennt. **Wrapping** bzw. **Kapselung** lässt die inkompatiblen Altanwendungen mit Hilfe von bereits erprobte Arten von „Middleware“ (CORBA, DCOM, RMI, EJB u.a.) schnell und kostengünstig in modernere Anwendungen integrieren und somit die Legacy-Funktionalität für neuere Applikationen nutzen. Vgl.[HoffR]

Dabei lässt sich die Anwendungsintegration in vier Typen unterteilen, je nachdem in welcher Softwareschicht die Integration stattfinden soll:

- Die **Replikation** kopiert Teile einer Anwendung zu mehreren „Slaves“ zusätzlich zur Masteranwendung
- Die **Daten-Integration** dupliziert mit Hilfe von geeigneten Tools die Daten einer Anwendung zu einer oder mehreren anderen Anwendungen, um ihnen den Zugriff (nur lesend) auf diese Daten zu ermöglichen, z.B. durch Screen-Scraping.
- Die **Funktionsintegration** ruft Programmcode einer Anwendung direkt auf, um es einer anderen Anwendung zu übertragen und basiert auf Komponentenmodellen wie zum Beispiel Microsoft COM, CORBA, JavaBeans oder RMI, die selbst RPCs sind. Auf die Daten kann lesend und schreibend zugegriffen werden
- **Totale Anwendungsintegration** wird erst dadurch erreicht, wenn eine Anwendung im Gefüge einer anderen Anwendung verfügbar gemacht wird, ohne die Anwendung dabei zu kopieren. Vgl.[Eich02]

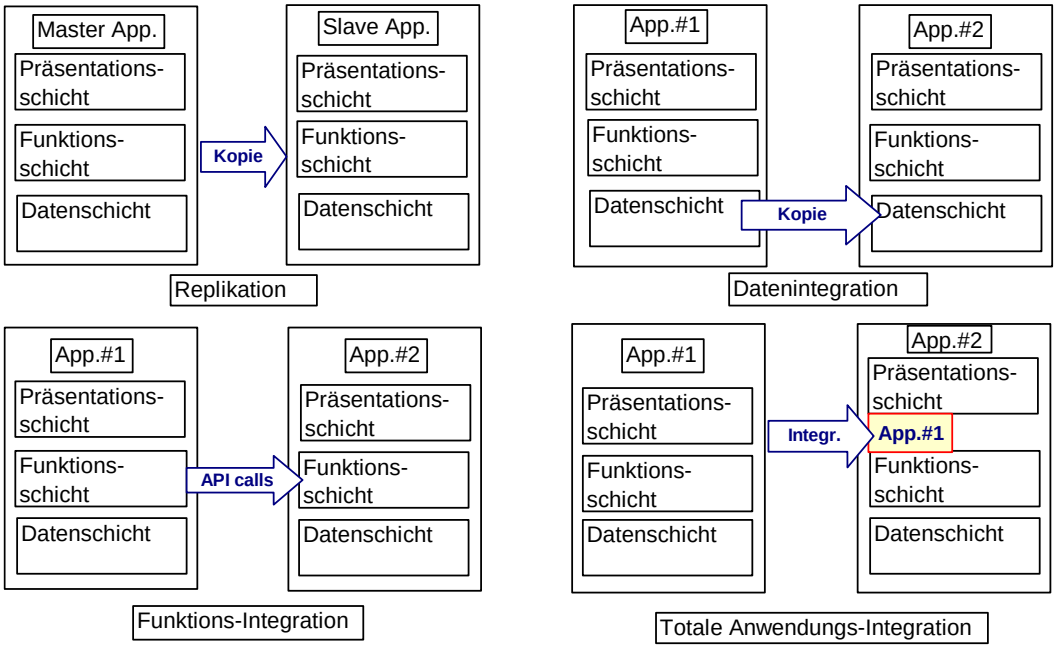


Abbildung 2 - Integrationstypen vgl. [Wong01]

Für eine unternehmensübergreifende E-Commerce (**B2B-und B2C**) muss auch der Datenaustausch über Unternehmensgrenzen hinweg im Web ermöglicht werden. Die bisherigen Integrationsmethoden sind sehr arbeitsaufwendig und ihre Komplexität beinhaltet viele Schmerzpunkte mit Bruchstellen und Abhängigkeiten, die eine sorgfältige Instandhaltung erfordern. Standards wie EDIFACT erfüllen die wirklich universelle, plattformunabhängige Kommunikation und den Datenaustausch bei Weitem nicht.

2.5 INTEGRATION MIT WEB SERVICES

Durch die Einführung von **Webservices-Technologien** können schlagartig viele Beschränkungen früherer Standards übersprungen werden. Sie eliminieren einen großen Teil des Integrationsbedarfs von Systemen untereinander. Obwohl die Entwickler weiterhin die Fähigkeiten der beteiligten Systeme entdecken müssen, wird der Verbindungsprozess stark vereinfacht und die Problempunkte der Integration – unterschiedliche APIs und die damit verbundenen Integrationsprobleme – wurden entfernt. Auch die firmeninterne EAI lässt sich mit Webservice-Techniken vorzüglich gestalten. Wichtig ist außerdem die Tatsache, dass die Integration neu konfiguriert werden kann, ohne das System teuer neu aufzubauen.
Vgl. [Gümüs]

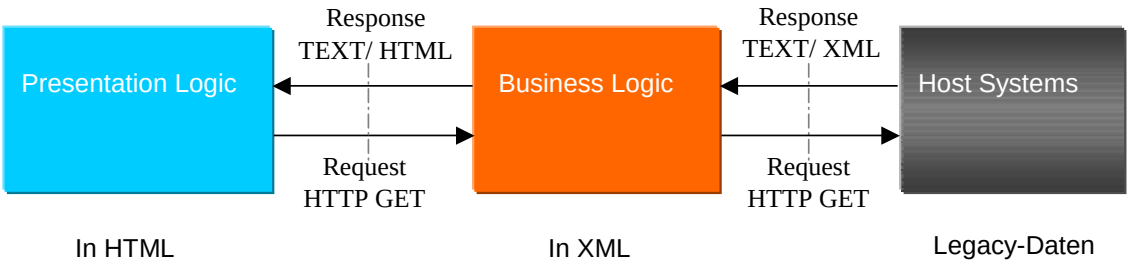


Abbildung 3 - Flussdiagramm der Legacy-Daten-Integration

Wrapping mit XML, XSL, SOAP

Mit XML ist es möglich beliebige Altanwendungen in neuere Applikationen zu integrieren. Das kann entweder dadurch passieren, dass die neue Applikation über XML-Dateien Daten mit der Altanwendung austauscht, oder dass mittels XML-RPC oder SOAP die Prozeduren der **L**egacy **I**nformation **S**ystem (LIS) aufgerufen werden. [Gümüs]

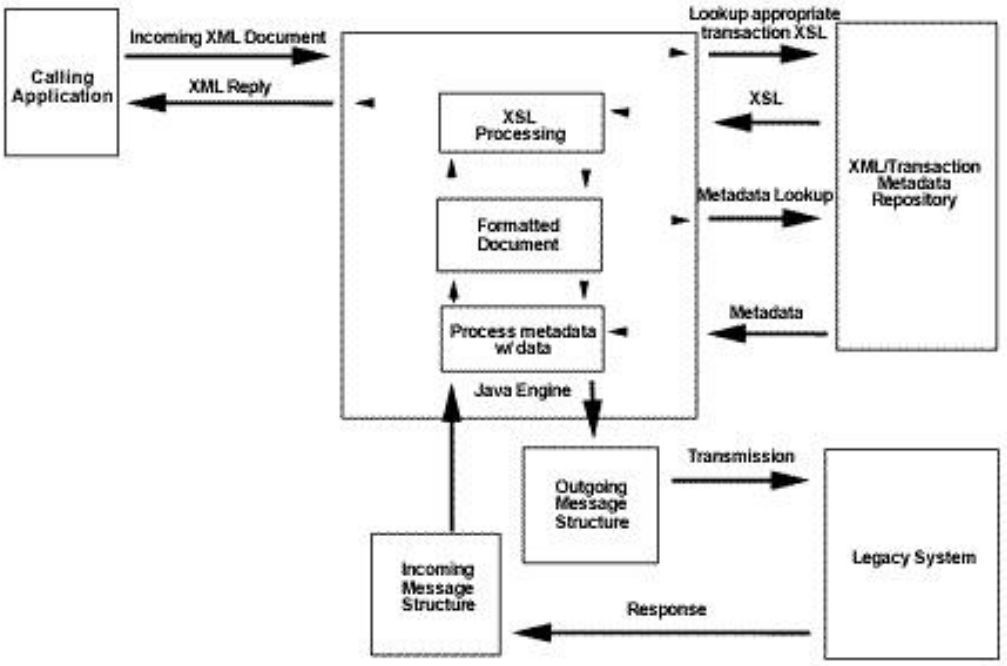


Abbildung 4 - Wrapping mit XML und XSL [Gümüs]

Mit den im Moment verfügbaren WS-Techniken lässt sich allerdings nur eine Integration auf funktionaler Ebene erreichen. Die wirkliche Anwendungsintegration scheitert aber noch z.B. an fehlender Unterstützung für Transaktionen oder der fehlenden Möglichkeit, ein Benutzer-Interface zusammen mit den zum Zugriff auf eine Anwendung notwendigen Daten (wie z.B. Schnittstellenbeschreibungen) in einer Service-Beschreibung zu kapseln.

Durch Wrapping wird es schließlich möglich sein, eine Anwendung innerhalb einer Anderen verfügbar (und für den Nutzer zugreifbar) zu machen, ohne Teile der Anwendung replizieren zu müssen. Die Arbeitsgruppe Web Services User Interface (WSUI, http://www.wsui.org) beschäftigt sich mit dieser Thematik. Sie benutzt ein einfaches Schema zur Beschreibung einer WSUI-Komponente, die über SOAP Webservices aufruft und XSLT Stylesheets dazu verwendet, eine Benutzersicht auf den Webservice zu erzeugen und so die Interaktion zwischen Benutzer und Service zu ermöglichen. Auch von IBM existiert eine neue Spezifikation, die sich mit einer ähnlichen Thematik beschäftigt, die Web Services Experience Language (WSXL, http://www106.ibm.com/developerworks/library/ws-wsxl/index.html), die es ermöglichen soll, Webservices als Komponenten über verschiedene Wege zu nutzen. So soll die direkte Nutzung eines Webservice über einen Browser ermöglicht werden, die indirekte Nutzung durch Einbindung des Webservice in ein Portal bzw. die Einbettung des Webservice in andere Anwendungen.

Allerdings befinden sich die Arbeiten noch in einem frühen Stadium, die man in http://www.xmethods.com/ aktuell verfolgen kann.

2.6 TRANSAKTIONEN BEI LEGACY-SYSTEMEN

Ein Programm wird als *Transaktionsprogramm* und eine aus seiner Ausführung resultierende als *Transaktion* bezeichnet, wenn das zugrundeliegende Laufzeitsystem dafür bestimmte Eigenschaften gewährleistet. Die Transaktion ist eine Folge von Aktionen eines ausgeführten Programms.

Transaktionssysteme basieren auf verteilte Client-Server-Anwendungen. Für den Anwender sind es parallele Prozesse und aus der Sicht der Programmierer atomare, konsistente, dauerhaft und untereinander isolierte ACID-Systeme.

ACID:

Die grundlegenden Eigenschaften einer Transaktion wie ***A****tomarität*, ***C****onsistenz*, ***I****solation* und ***D****auerhaftigkeit* sind wie folgt:

Atomarität (Atomicity):

Eine Transaktion wird als eine unteilbare Aktion angesehen. Sie läuft entweder gar nicht ab oder sie wird komplett ausgeführt. Bei räumlich verteilten Transaktionen müssen entweder Änderungen an allen beteiligten Ressourcen gleichzeitig oder an keiner durchgeführt werden. Teilausführungen sind verboten.

Konsistenz (Consistency):

Eine Transaktion überführt einen Datenspeicher von einem konsistenten in einen anderen konsistenten Zustand. Nur während der Ausführung einer Transaktion darf der Zustand kurzzeitig inkonsistent sein.

Isolation (Isolation):

Eine Transaktion muss unabhängig von anderen eventuell im System laufenden Transaktionen sein. Interaktionen sind nicht erlaubt, sie müssen also zumindest scheinbar seriell ablaufen.

Dauerhaftigkeit (Durability):

Nach Beendigung oder Bestätigung einer Transaktion müssen alle Änderungen erhalten bleiben. Dies gilt auch für den Fall eines Systemausfalls: Die an einer Transaktion beteiligten Ressource-Manager (wie Datenbanken o.ä.) müssen in der Lage sein, Änderungen wieder rückgängig zu machen, und sie müssen so stabil sein, dass sie nach einem Neustart selbst wieder in einen konsistenten Zustand versetzen können.

Anwendungsfälle:

Es gibt bestimmte Fälle, die mit Hilfe von Transaktionsprogrammen folgende Anwendungen verknüpfen:

- OLTP -Anwendung (On-Line Transaction Processing)
- E- Commerce
- Workflow-Anwendung

Bei Webservices können die Transaktionen zur „Orchestrierung“ von komplexeren Diensten aus einzelnen WS zu verteilten Anwendungen kombiniert werden. Ergo umfasst eine WS Transaktion das Zusammenspiel mehrerer WS, die über das Internet kommunizieren müssen. Hierbei müssen gemeinsame Standards definiert werden, um die bisherige getrennte Bemühungen ,wie z.B. Microsofts XLANG oder WSFL von IBM, zu konsentieren.
So soll BPEL, ein gemeinsamer Vorschlag von BEA, IBM und Microsoft, erlauben, Geschäftsprozesse zu definieren und dabei WS zu nutzen und komplexe Abläufe auch wieder als WS für andere Anwendungen bereitzustellen. BPEL wird zur Zeit beim W3C heiß diskutiert.Vgl.[c’t02]

3. GRUNDLAGEN DER WEBSERVICES

3.1 DEFINITION

Obwohl die heutige Web-Technologie bereits die Basis vieler Systeme bildet, wird sie in erster Linie nur zum Informationsabruf eingesetzt. Unzählige Dienste tummeln sich um bestimmte Plattformen herum. Sie sind stark abhängig von deren Spezifikationen, die als technologische Hürden jedoch Verbindungen zu anderen Plattformen hemmen.

Neue Geschäftsanforderungen zwingen zur Bildung globaler Standards, um system-unabhängig mit allen Plattformen kommunizieren zu können. Insel-Lösungen, um ein Unternehmen mit anderen Unternehmen, Extranet oder Marktplatz zu verbinden, sollten bald Geschichte sein. Dazu wurden von namhaften Softwareschmieden wie Ariba, IBM, Microsoft, Sun u.a. nach und nach eine Reihe von Standards vorgeschlagen, die allesamt Bausteine der heutigen Webservices sind. Zuerst einigte man sich auf die Beschreibungssprachen DTD und XML, dann folgten das Internetprotokoll SOAP, WSDL für die Beschreibung der Service und UDDI als zentrales Register, womit schon die Architektur des Webservices grob vorgezeichnet ist. Aber was sind nun Webservices eigentlich ? Es existieren viele ähnliche Definitionen, wie etwa :

„Ein Web-Service ist eine Komponente, die ihre Funktionalität über eine veröffentlichte Schnittstelle anbietet und über ein offenes, im Internet verwendetes Protokoll zugreifbar ist.“ [Jeckle]

Definition á la W3C:

“A Web service is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML based messages exchanged via internet-based protocols. [WSw3c]

Eigene Definition:

Web Services sind lose gekoppelte Softwarekomponenten, die spezifische Funktionalität kapseln und via standardisierte Internetprotokolle zugänglich sind.

Durch die lose Koppelung der Dienste erlangt man mehr Flexibilität und reduziert so den Kommunikationsaufwand.

Die vollständige Kapselung von Komponenten macht sie ohne zusätzlichen Implementationsaufwand innerhalb einer anderen Anwendung verfügbar.

Die Web Service Infrastruktur ermöglicht den Zugriff auf die Dienste über harmonisierte Kommunikations -und Transportprotokolle.

3.2 VORTEILE

Eine Schlüsseleigenschaft und das bisherige Erfolgskonzept des Internet liegt darin, dass Standards wegen der verteilten Architektur des Webs eher auf Protokollen basieren als auf Implementierungen. Es setzt sich aus ganz heterogenen Technologien zusammen, die über gemeinsame Protokolle erfolgreich miteinander kommunizieren und nicht über irgendeine Middleware. Kein Softwarehersteller ist in der Lage, im Internet einen eigenen Standard durchzusetzen. Keine Programmiertechnik kann als einzige Lösung das Internet beherrschen. Vgl.[Ecinweb]

Hier setzen Webservices an, die das enorme Potenzial des Internet erst richtig ausnutzen.

- Unabhängige Implementation und Funktionalität von Hard- und Software-Plattformen.
- Kommunikation zwischen Anwendungen über Standard Internet-Protokolle in Echtzeitnähe
- Lose gekoppelte Softwarekomponenten minimieren den Kommunikationsaufwand und ermöglichen Plus an Flexibilität
- Legacy Systeme können standardisiert, permanent und wiederverwendbar gekapselt werden
- Automation von Geschäftsprozessen
- Erhöhung der Produktivität und der Effektivität von Anwendungen ohne Mehrinvestitionen

Die Produktivität und die Effektivität der Geschäftsprozesse lässt sich erhöhen, denn Webservices verursachen bedeutend weniger Investitionsausgaben durch Einbeziehen der vorhandenen Systeme und durch Verringerung der Entwicklungszeit für Anwendungen. Damit wahrt die Firma die Chance, für Kunden, Partner und Lieferanten zugänglichen Dienstleistungen kurzfristig und den Anforderungen entsprechend anzubieten.

Plattformneutralität und Interaktivität von Anwendungen über Standard Internet-Protokolle:

Applikationen können unabhängig von ihren Protokollen, Sprache oder Plattform miteinander kommunizieren. Ein Webservice kann von einem anderen Service aufgerufen werden und dessen Funktionen nutzen, als wäre es sein eigenes Modul. Das funktioniert, weil die Schnittstelle jedes Services in einem Standard gestaltet und definiert ist, so dass Dritte ohne Kenntnis der dahinterliegenden (da verkapselten) Software-Infrastruktur den Service aufrufen können. Dabei erübrigt sich für Business Anwendungen die leidige Frage nach der Integrationssoftware der so genannten Middleware zwischen zwei verschiedenen Systemen (z.B. SAP, Oracle, Sun, HP, Bea, Microsoft etc.), die somit universell gelöst werden kann. Manager können sich so endlich auf die eigentlichen Herausforderungen der Geschäftslogik konzentrieren. Vgl.[Ecinweb]

Anders als Web-Seiten und Desktop-Anwendungen repräsentieren die Webservicesyntax Beziehungen sowohl von Mensch zu Maschine als auch Maschine zu Maschine und interagieren nahezu in Echtzeit.

Lose gekoppelte Softwarekomponenten durch vollständige Kapselung von spezifischer Legacy-Funktionalität:

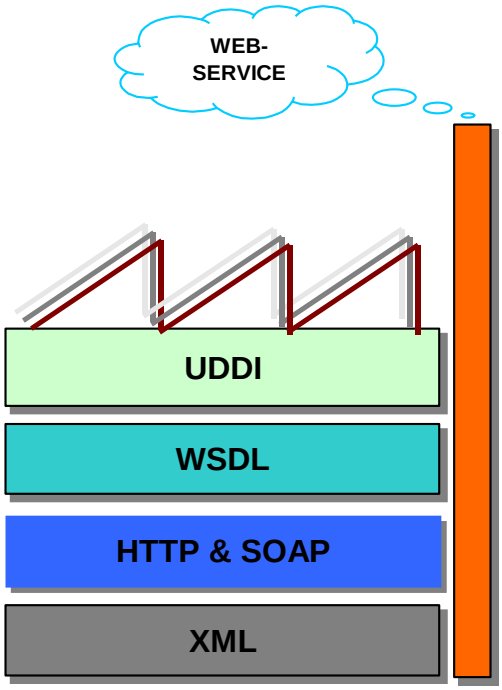
In traditionellen Anwendungen muss der Entwickler gänzlich das System verstehen, um eine Komponente aus einer festen Verbindung von Tochterelementen zu extrahieren und durch andere zu ersetzen, die er durch Web Services nicht braucht: Eine Anwendung kann innerhalb einer anderen dargestellt werden, ohne die Teile der Anwendung repliziert werden müssen. Semantisch getrennte Funktionalität wird eingekapselt, deren Schnittstellen und Implementierung werden entkoppelt.

Geschäftprozess-Automation

Web Services erleichtern e-business. Durch den Einsatz der Web Services entfällt die Notwendigkeit, unterschiedliche Integrationen zwischen den Partnern, Kunden oder Lieferanten durchzuführen. So kann man einheitliche, funktionelle EAI (Enterprise Application Integration) bilden und B2B Transaktionen erleichtern.

Sie helfen den Anwendern, sich auf seine Kernkompetenzen zu konzentrieren.

3.3 ARCHITEKTURKONZEPTE UND FUNKTIONSCHEMA



WS besteht aus folgenden Grund - Bausteinen:

- XML (Extensible Markup Language)
- SOAP (Simple Object Access Protocol)
- WSDL (Web Services Description Language)
- UDDI (Universal Description, Discovery and Integration)

Abbildung 5 - Prinzipielle Architektur der WS

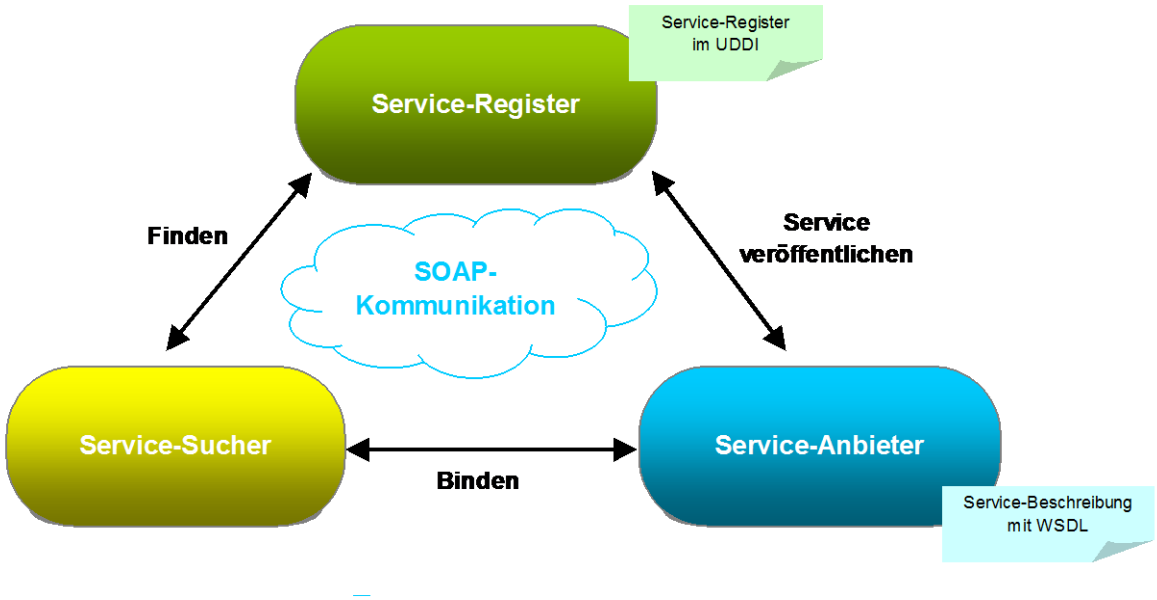


Abbildung 6 - Webservice Funktionsschema, Vgl.[ibmfkt]

Web Service Architektur zerfällt in (mindestens) drei Rollen:

Der **Dienstanbieter** übernimmt die Implementierung, Betrieb und Wartung von ihm über Web bereitgestellten Dienste.

Der **Dienstnachfrager** sucht angebotene Dienste nach seinen Kriterien aus und integriert sie in eigene Dienste und Applikationen.

Der **Dienstmakler** bietet Speicherung von Diensten an und verwaltet sie. Er ermöglicht außerdem das automatische Auffinden existierender Dienste. Vgl.[Jeckle02]

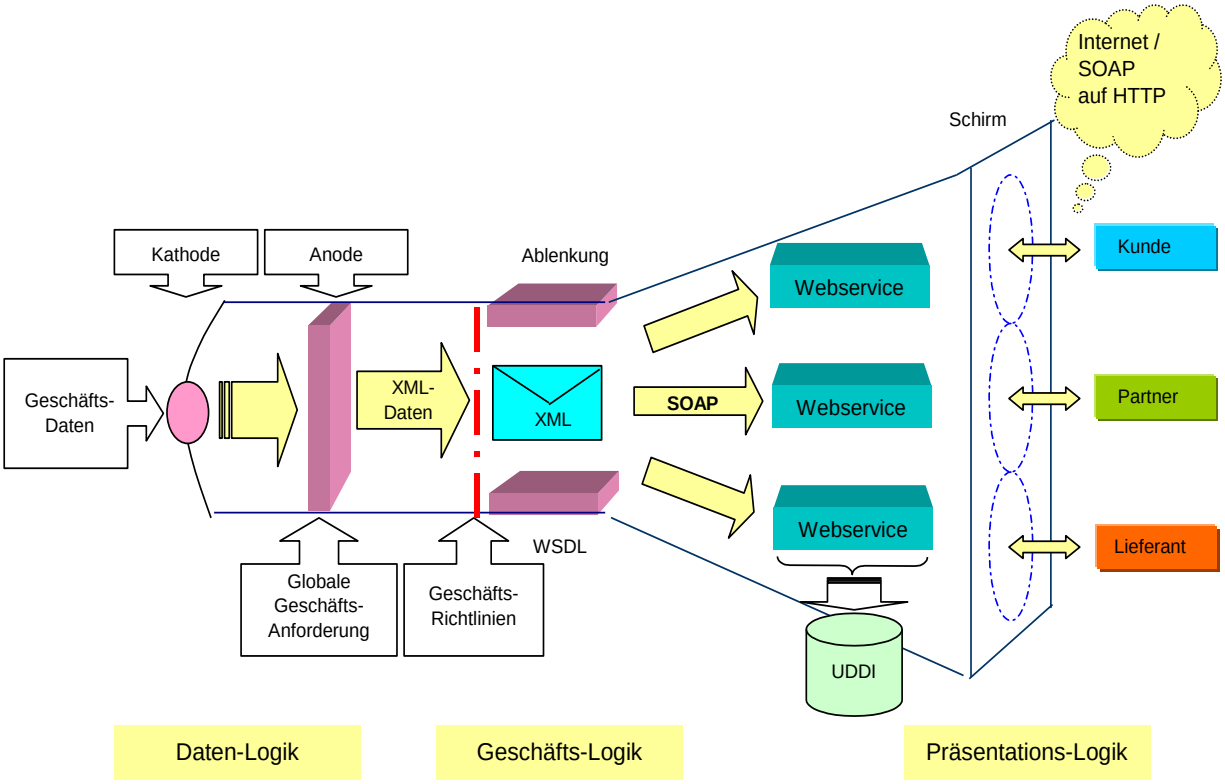


Abbildung 7 - Funktionale Parallelität zwischen Braunschen-Röhre und Webservice

3.4 SPEZIFIKATIONEN

Bei der Implementierung der WS werden folgende Tools verwendet:

- XML - Extensible Markup Language
- SOAP - Simple Object Access Protocol
- WSDL - Web Services Description Language
- UDDI - Universal Description, Discovery and Integration

3.4.1 XML

XML als „Treibstoff“ für Anwendungen ist eine Metasprache für die Definition von Datenobjekten, d.h. sie liefert die Regeln, um Textformate für strukturierten Inhalt zu erstellen und beschreibt das Verhalten von Computer-Programmen, die solche Dokumente verarbeiten. Der Vorteil bei XML liegt darin, dass die Daten relativ einfach zu lesen, zu erstellen und leicht erweiterbar sind. Durch zusätzliche Ableger wie XSL / XSLT können XML-Dateien zudem in jedes andere Format umgewandelt und transformiert werden und eigenen sich hervorragend für die Daten-und Prozess-Integration. Diese neutralen Eigenschaften machen sie für den Aufbau der WS unentbehrlich.
Ausführliche Beschreibung der Regeln und Funktionsweisen würde den Rahmen dieser Arbeit sprengen, der Hinweis auf XML- Spezifikationen von W3C, Vgl.[xmlw3c] mag genügen.

3.4.2 XSD

XML kann der Schlüssel zu System-Interoperabilität sein, aber eine erfolgreiche Integration erfordert eine präzise Definition des Inhalts der XML-Daten. Mehrere Mechanismen bieten diese Funktionalität, unter denen die Schemata nach der XML Schema Definition (XSD) eine führende Stellung einnehmen. Grundstrukturen wie .Net unterstützen verschiedene Schemafunktionen, z. B. die Validierung von XML-Daten gegenüber einem XSD-Schema. Vgl.[zdnet]

3.4.3 XSL & XSLT

Die Hauptaufgabe der XML ist die Beschreibung der Daten in allgemeingültiger Form. Bei der Darstellung z.B. auf dem Bildschirm fehlen jedoch die Formatierungsangaben. Die Extensible stylesheet language XSL spezifiziert hierbei die Formatierung und bestimmt das Layout eines XML-Dokumentes, um unter Benutzung von XSL-Transformation zu beschreiben ,wie der Inhalt von XML- Dateien in beliebiger Form, wie z.B. in HTML, WML, dargestellt werden kann. Vgl.[XSLT] Dazu wurden während der Entwicklung der XML-Spezifikationen durch die W3C-Arbeitsgruppe die XSL und die XSLT vorgeschlagen.

XSLT kann in einigen Fällen auch unabhängig von XSL benutzt werden, obwohl sie nicht als vollständig universelle XML- Transformationssprache entworfen wurde. Primär ist sie jedoch für Transformationen vorgesehen, wenn XSLT als Bestandteil von XSL benutzt wird. Vgl.[Klute]

3.4.4 SOAP

Das **Simple Object Access Protocoll** (SOAP) stellt als W3C-**Standard-Kommunikationsprotokoll** einen einfachen und durchsichtigen Mechanismus zum Austausch von **Daten und Dokumenten** zwischen Rechnern im Web zur Verfügung. Bei seinem Entwurf wurde bewusst auf komplexe Mechanismen eines verteilten objekt-orientierten Systems verzichtet, es fügt sich in die bewährten Internetprotokolle wie HTTP, SMTP nahtlos ein. Durch die Verwendung von XML als Codierung bleibt SOAP deshalb einfach, flexibel und erweiterbar, es arbeitet unabhängig von Hardware, Betriebsystem, Sprache, Implementierung oder Softwarehersteller.

Das Arbeitsprinzip von SOAP ist sehr einfach: Z.B. wird der HTTP-Header um SOAP-spezifische Einträge erweitert und an diesen die XML-Nachricht angehängt. Es kann somit jeden HTTP-Server oder Firewall leicht durchdringen.

SOAP besteht neben dem Header aus einem »Envelope« und enthält darin den »Body«. Innerhalb des Bodies befindet sich wiederum die Anfrage beziehungsweise Antwort an das empfangende System, das den SOAP-Call verstehen und vor allem die darin enthaltene XML-Nachricht interpretieren muss. Dessen SOAP-Response enthält die Antwort darauf. Vgl.[ntwk02]

SOAP wird durch drei Komponenten beschrieben: Vgl.[iicm01]

- **SOAP Envelope** definiert die Attribute einer Nachricht, z.B. Inhalt und Adressat der Nachricht, ob bindend oder optional
- **SOAP encoding** beschreibt die übertragenen Daten in Hinblick auf Serialisierung und Kodierung
- **SOAP RPC representation** definiert die Darstellung von RPC's (Remote Procedure Call) und deren Antworten

SOAP beschreibt außerdem die Schnittstelle eines **WS-Objekts** mit sämtlichen **Methoden**, **Parametern** und **Rückgabewerten**. Für die Implementierung können bereits dutzende verschiedener SOAP-Toolkits für alle Arten von Programmiersprache und Betriebssystem benutzt werden.
Die in dieser Arbeit angewandte **.NET** Architektur von Microsoft z.B. benutzt SOAP als zentrales Protokoll.

Weil die Kommunikation mit SOAP innerhalb wie außerhalb eines Unternehmens gleichermaßen gut funktionieren kann, besteht ein gewisser Vorteil gegenüber existierenden Architekturen wie CORBA oder DCOM.

Die unterschiedlichen, lokal verteilten Anwendungssysteme eines durch Zukäufe, Fusionen oder wie auch immer wachsenden Unternehmen können über SOAP-Wrapper gekapselt werden und mit der Kombination von WSDL die Integration (EAI) ermöglichen.

3.4.5 WSDL

Als IBM und Microsoft sich anschickten, SOAP für WS-Anwendungen einzuspannen, erkannten sie schnell ein Manko: Für die Implementierung eines Services sind genauere Informationen zu den SOAP Nachrichten, wie etwa das Format, nötig.

Da die dazu entwickelten Tools inkompatibel waren, wurde ein gemeinsames Protokoll notwendig; die Web Service Description Language WSDL soll nun ein WS komplett mit allen benutzten Datentypen, Protokollen, Nachrichtenformate, URLs beschreiben können, ohne ihn zu implementieren. Der so selbstbeschriebene WS kann von außen plattformunabhängig genutzt werden, der Client braucht keinerlei Details zu Plattform oder Implementierungsart.

Die bloße Definition bezieht sich auf folgende Eigenschaften von Daten:

- Schnittstellen-Informationen,
- Datenartinformationen für alle Nachrichten (Request und Response),
- Transportprotokoll-Informationen,
- Bezugs- Informationen für das Lokalisieren des spezifizierten Services

Mit Hilfe der WSDL kann man einen Webservice lokalisieren und jede seiner publikativen Funktionen hervorrufen und sogar mit WSDL-Tools diesen Prozess automatisieren.

Die WSDL Spezifikationen:

Ein WSDL- Dokument beschreibt einen Service mit folgenden Elementen:

- *Type,* Datentypdefinitionen, z.B. unter Verwendung von XSD
- *Operation,* Eine abstrakte Beschreibung einer Serviceleistung.
- *Message,* Definition von Datenkommunikation.
- *Port Type,* Eine abstrakte Sammlung von Ausführungen, die von einer bzw. mehrerer Schnittstellen unterstützt werden.
- *Binding,* Die Beschreibung der Datenformatspezifikation für einen bestimmten *PORT TYPE*.
- *Port,* Eine einzelne Schnittstellendefinition bestehend aus *BINDING* und Netzwerk-adresse.
- *Service,* Eine Sammlung von verbundenen Schnittstellen.

3.4.6 UDDI

Der nunmehr ansetzende Run auf die Fülle von unterschiedlichsten Webservices wirft die Frage auf, wie jeder denkbare Webservice eindeutig nach Eigenschaften koordiniert und lokalisiert werden kann. Eigens dafür vorgeschlagene **U**niversal **D**escription, **D**iscovery and **I**ntegration definiert ein Dienstverzeichnis, in dem Anbieter ihre Services registrieren und Interessenten sie gezielt suchen können. Webservices sind im UDDI Register immer abhängig von geschäftlichen Anwendungen. Analog dazu enthält es eine Kategorisierung nach Branchen und Unternehmen. Vgl. [Ecinweb]

Ein UDDI-Verzeichnis-Struktur sieht zum Beispiel so aus:

Seite	Operation	Information
Weiss	veröffentlichen	Geschäftlich
Gelb	suchen	Service
Grün	verbinden	Verbindungsdaten Service Spezifikation Details

Abbildung 8 - UDDI – Verzeichnisstruktur

Mit Hilfe der Webservice-Konzepte kann man drei Arten von Informationen in ein UDDI Register eingeben:

Weiße Seite

Sie ermöglicht grundlegenden Kontakt zu den Firmen-Informationen, einschließlich des Handelsnamens, Adresse, Kontaktinformationen, etc. Diese Informationen lassen sich von Anderen im Webservice nach Ihrer Geschäftskennzeichnung aufdecken.

Gelbe Seite

Darin werden Informationen ähnlich dem Telefonbranchenbuch in unterschiedlichen Kategorisierungen beschrieben. Diese lassen sich durch Webservices auffinden, sobald ihre Branche beschrieben wird. Also ein "Who is Who“, ein Verzeichniseintrag für Unternehmen.

Grüne Seite

Technische Informationen, die das Verhalten und die gestützten Funktionen eines Web services beschreiben, die durch eine Organisation unterhalten wird. Diese Informationen schließen die Bündelung der Informationen von Webservices mit ein und zeigen, wo die Webservices lokalisiert sind.

Wie wird UDDI verwendet ?

UDDI hat verschiedene Gebrauchsmöglichkeiten. Von der Perspektive eines Geschäfts-prozesses ist UDDI ähnlich wie eine Internet-Suchmaschine. Es organisiert und registriert URLs für Webservices.

Hinsichtlich der Software-Entwicklung verwendet man den UDDI, um Web Services zu veröffentlichen und in den Registern nach ihnen zu suchen. Es ist denkbar, dass Software bald Services dynamisch ohne menschliches Zutun entdeckt und sie verarbeitet.

4. SICHERHEITSASPEKTE

Die Sicherheit spielt in der Anwendungsintegration eine große Rolle. Beim Austausch sensibler Daten kann bereits innerhalb eines Unternehmens nicht auf Sicherheitsmaßnahmen wie Verschlüsselung verzichtet werden. Für die Business-to-Business Integration über Unternehmensgrenzen hinweg ergeben sich noch weitergehende Anforderungen.

Es müssen also Möglichkeiten gefunden werden, Webservices mit Zugriffskontrollen zu versehen, damit eine ganze Reihe von Sicherheitsfunktionen erfüllt werden können. Nachfolgend eine Auflistung der nach [Colan01] für Webservices relevanten Anforderungen im Sicherheitsbereich:

- *Identifikation*: Jeder Anbieter und jeder Nutzer muss die Identität des jeweils anderen kennen.
- *Authentifizierung*: Identitäten müssen zweifelsfrei nachweisbar sein.
- *Autorisierung*: Bestimmte Webservices sollen nur von bestimmten Anwendern genutzt werden können.
- *Datenintegrität*: Die gesendeten bzw. empfangenen Daten dürfen auf dem Weg vom Sender zum Empfänger nicht unberechtigt verändert werden können.
- *Vertraulichkeit*: Es muss sichergestellt werden können, dass niemand ausgetauschte Informationen mitlesen kann, z.B. durch den Einsatz von Verschlüsselungstechniken.
- *Protokollierung*: Ausgetauschte Informationen müssen bei Bedarf mitprotokolliert werden können, damit auch im Nachhinein die Möglichkeit besteht, erfolgte Kommunikation nachvollziehen zu können.
- *Nachweisbarkeit* (Non-Repudiation): Sowohl der Sender als auch der Empfänger einer Nachricht müssen zweifelsfrei nachweisen können, dass erstens der Sender die Nachricht auch wirklich verschickt hat, und dass zweitens der Empfänger die gleiche Nachricht auch wirklich erhalten hat. [EichSec]

SOAP-DSIG (W3C Note) XML Signature Specification (W3C Recommendation)	Digitale Signatur von XML Signature Specification basierende SOAP-Nachrichten erlaubt **Identifikation** und **Authentifizierung** des Verfassers einer Nachricht. Gewährleistet die **Integrität** der übermittelten Daten und kann zur **Non-Repudiation** verwendet werden. Die Identität kann wiederum zur **Autorisierung** herangezogen werden.
XML Encryption for Syntax & Processing (W3C Recommendation)	Zur Verschlüsselung von XML Dokumenten und SOAP Nachrichten mit Gewährleistung von **Vertraulichkeit**. So verschlüsselte Nachrichten können auch verschlüsselt **mitprotokolliert** werden.
XKMS (W3C Note)	Registrieren und Auffinden von öffentlichen Schlüsseln, wie sie z.B. für die digitale Signatur o. Verschlüsselung notwendig sind.
SAML (Oasis Working Draft,V 1.0)	Austausch von sicherheitsrelevanten Daten wie Authentifizierungs-u. Autorisierungs-informationen.
XACML (OASIS Working Draft)	Zugriffskontrolle z.B. für separate Elemente von XML Dokumenten

Abbildung 9 - Zusammengefasste Sicherheitsstandards

Mit Hilfe der o.a. Sicherheitsstandards kann die Implementierung sicherer, interoperabler Webservices ermöglicht (SOAP-DSIG, XML Encryption) oder vereinfacht (XKMS, SAML, XACML) werden. Schon allein an der Vielzahl der Bemühungen in diesem Bereich lässt sich die Bedeutung der Sicherheit für die weitere Verbreitung von Webservices erkennen.

XML soll dabei als wichtigste Schnittstelle zwischen modernen Web- und traditionellen Transaktions-Anwendungen vermitteln und die bislang zwischen den Welten getrennten Dienste Authentisierung und Autorisierung miteinander koppeln. Die Entscheidung für das sich zur Universalsprache mausernde XML fällt nicht überraschend, da doch führende Marktanalysten davon ausgehen, dass künftig alle E-Commerce-Anwendungen und Management-Applikationen ihre Daten per XML austauschen und so nahtlos miteinander kommunizieren können. Jetzt müssen die Grundlagen erarbeitet werden, um die bislang nicht berücksichtigten proprietären Datenformate der Legacy-Transaktionsanwendungen an die XML-Merkmale anzupassen. Da XML nicht objektorientiert, sondern übersetzend und vermittelnd zwischen diesen Applikationen arbeitet, kann die Sprache jede Datenstruktur

darstellen, unabhängig vom vorliegenden Format. Der künftige Standard muss sich dazu verpflichten, die Datenintegrität während des Transfers zu garantieren und die zu übermittelnden Daten an die jeweiligen Applikationen anzupassen.

Das World Wide Web Consortium (W3C) hat erst kürzlich zwei neue Empfehlungen für XML Encryption Syntax and Processing Vgl.[Core02] und Decryption Transform for XML Signature Vgl.[Decr02] veröffentlicht. Diese beiden Protokolle sollen die Übermittlung von vertraulichen Daten durch Web-Seiten und Web Services sicherer gestalten. Im Unterschied zu den bereits gängigen Verfahren zur Verschlüsselung von XML-Dokumenten erlauben die beiden neuen Protokolle laut W3C eine Verschlüsselung von ausgewählten Passagen oder Elementen eines Dokumentes, beispielsweise eine Kreditkartennummer, die in ein XML-Formular eingegeben wird.

Viele Standards befinden sich aber noch in der Entwicklung, so dass sich sicherlich noch einige Änderungen ergeben werden, bevor die endgültigen Versionen verfügbar und damit auch außerhalb von Testprojekten einsetzbar sind. Dennoch wird aber bereits deutlich, dass nur durch gemeinsame Standards für die Implementierung von Webservices mit Unterstützung von Verschlüsselung oder digitalen Signaturen WS überhaupt für die Integration über Unternehmensgrenzen hinweg sicher einsetzbar wird.

5. MICROSOFT .NET PLATTFORM

Vgl [MS.Net]

MS.NET Ist ein leistungsfähiges Plattform für die Entwicklung und Einsatz von objekt-orientierten, verteilten Anwendungen. Durch eine universelle Ausführungsumgebung für viele Programmiersprachen unterstützt es viele namhafte Programmiersprachen wie z.B. C++, J-Script, VB.NET, Java, Perl, einschließlich COBOL, Eiffel, LISP, Python u.a. und ermöglicht neben anderen Anwendungen auch Webservices zu verwirklichen.

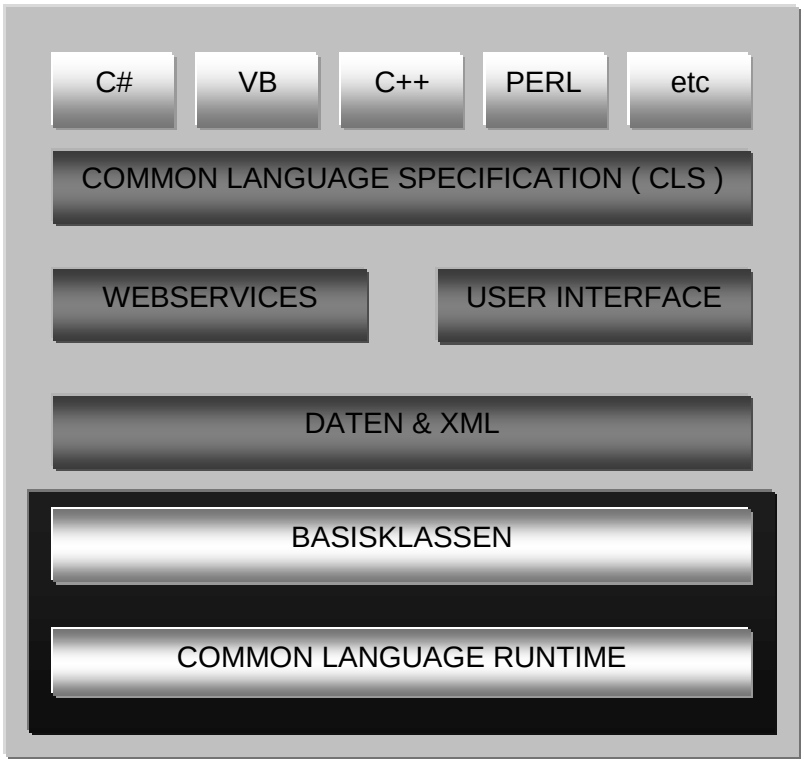


Abbildung 10 - MS.NET Architektur

Seine wichtigste Errungenschaft ist sicherlich die Laufzeitumgebung **C**ommon **L**anguage **R**untime, die besondere Dienste in Form von Basisklassen, Typprüfung, COM Marshalling, Programmierungssupport mit Threads, Sicherheitsmechanismen und vieles andere mehr bietet. Ein Garbage-Collector vereinfacht und automatisiert die Speicherverwaltung.

Alle unterstützten Sprachen werden mit Hilfe der CLR über einen binären Zwischencode MSIL (Microsoft Intermediate Language) kompiliert, der von den Modulen und Betriebssystemen unabhängig ist. Die Übersetzung wird just in time durchgeführt. Diese Eigenschaft ermöglicht eine losere Kopplung zwischen dem Betriebssystem und Anwendungsprogrammen und prädestiniert die .NET besonders für die sprach-und plattformunabhängige Implementierungen.

CLR zeigt folgende Eigenschaften:

- Ein gemeinsames Typsystem (CTS) bietet ein Konzept für Datentypen an, das von der speziellen Programmiersprache unabhängig ist.
- Ein JIT- Compiler übersetzt den MSIL-Code für die vorliegende Zielmachine und optimiert ihn bei der Gelegenheit. Dadurch läuft jede unterstützte Programmcode im .NET immer als Maschinensprache.

Microsoft stellt .NET auf der Entwickler-Website folgendermaßen vor:

Definition .NET: Microsoft .NET is an XML Web services platform that will enable developers to create programs that transcend device boundaries and fully harness the connectivity of the Internet.

Es bietet folgende Eigenschaften, die sich direkt mit Webservices befasst:

- Bildung eines kompletten Rahmens für einen sprachunabhängigen Bibliotheken
- Hilfe bei der Selbstbeschreibung von Softwarebausteinen
- Unterstützung der Multi-Sprach-Integration
- Anbieten einer neuen Technik, Windows-Desktop-Anwendungen mit den Windows-Formkategorien zu entwickeln
- Die ADO .NET-Module stellen eine neue getrennte Architektur für Datenzugang über das Internet zur Verfügung
- Das .NET-Framework und der CLR bauen auf Standards wie XML auf und sind praktisch auf jeder Ebene erweiterbar.

Unterstützung der Webservices

Es bietet volle und flexible Unterstützung der Webservices von der Implementierung bis zur Entfaltung (Deployment).

Namensräume und Klassen:

Die .NET- Typen werden durch eine Hierarchie von Namen strukturiert, die durch Punkte getrennt werden, so dass verwandte Typen logisch gruppiert werden. Sie dienen zur übersichtlicheren Organisation der Framework-Klassen. Ein Namensraum fasst die zusammengehörigen Klassen zu logischen Gruppen zusammen.

Die wichtigsten Namensräume sind:

- System.Web
- System.Web.Services
- System.Web.Services.Description
- System.Web.Services.Discovery
- System.Web.Services.Protocols
- System.Web.Security

5.1 ASP.NET

Dahinter steckt ein neuer Ansatz, dynamische und interaktive Web- Anwendungen zu programmieren; denn ASP.NET ermöglicht die totale Trennung von Layout und Geschäfts-logik und fegt die Schwachpunkte des früheren MS Active Server Pages beiseite.

Die Basis für die neue Architektur ist die .NET-Laufzeitarchitektur CLR und eine enge Integration mit Microsofts Entwicklungsumgebung Visual Studio.

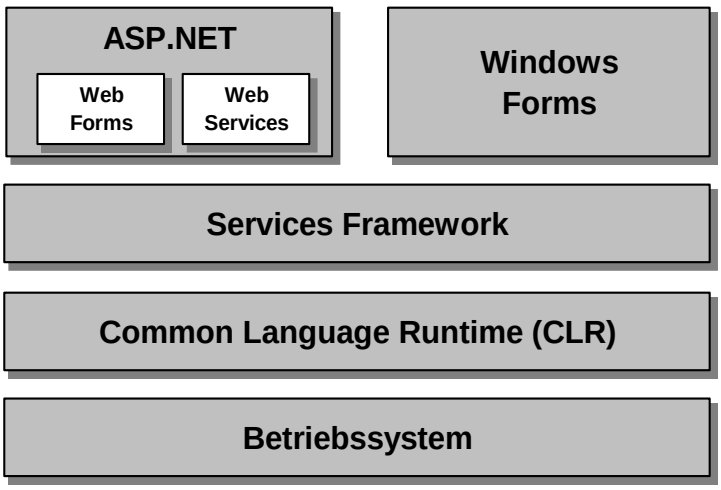


Abbildung 11 - ASP.NET-Architektur

Abbildung 11 zeigt die Schichten dieser Architektur. Auf die Schicht der Service Framework setzen sich weitere Form-Schichten auf, die es erlauben, Anwendungen je nach Funktion aufzubauen. Wenn man eine Windows- Client- Anwendung bauen möchte, bietet Visual Studio dafür Windows Forms an. Für Web-Anwendungen kann man ein Programmiermodell verwenden, das sich Web Forms nennt. Funktionsaufrufe aus dem HTML-Code und ähnliches können vor dem Benutzer vollkommen versteckt werden. Auch gibt es keinen Skript-File mehr, der zur Laufzeit jedes mal geparst und interpretiert wird, sondern Sourcecode, der mit einem Just-in-Time-Compiler übersetzt wird.

5.2 C#

Die Sprache C# stammt größtenteils von C beziehungsweise C++ ab, enthält allerdings auch Anlehnungen an Java. Dabei hat MS versucht, die Sprache so einfach wie möglich und damit weniger anfällig für Fehler zu machen. C# ist eine von über zwei Dutzend Compiler-Sprachen, die von ASP.NET klaglos verarbeitet werden. Sogar COBOL ist im Boot.

Die größte Gemeinsamkeit mit Java dürfte wohl sein, dass die von C# erzeugten Programme beinahe plattformunabhängig sind. Ihre Programme laufen nicht nur unter dem Betriebs-system, unter dem sie die Anwendung entwickelt haben, sondern auch auf anderen Plattformen, für die das .NET Framework existiert. Dabei braucht keine einzige Zeile Quellcode angepasst zu werden, denn Sprachen dieses Bundes können ineinander integriert werden. C# ist besser strukturiert als Visual Basic und leistungsfähiger als ihre C-Vorgänger oder Java, weil es APIs und Systeme benutzen kann, die nicht in der Javasprache entwickelt wurden. C# -Code wird direkt auf dem Prozessor ohne Interpretierung ausgeführt und bietet dadurch jederzeit maximum speed.

6. WEBSERVICES AM BEISPIEL DER TRANSAKTION PASSAGIER LISTE (NPL)

6.1 PROJEKTZIELE

Konzept: Die Nutzung von Transaktionen durch das Internet mittels eines Browsers ist durch den Hersteller (UNISYS) mit der Einführung eines Webservers auf dem Host ermöglicht worden (WebTS). Die Anpassung der Legacy Anwendung erlaubt den Schedule einer Transaktion durch einen nach dem HTTP-GET Protokoll formatierten URL.

Im Rahmen dieser Diplomarbeit soll nun eine ASP.Net basierende Anwendung erstellt werden, durch die neben der Autorisierung eine Auswahl von Transaktionen mit Hilfe eines Webservices über eine Standalone-Anwendung als auch über aktive Webpages ausgeführt werden können.

Für die Entwicklung sollte das Microsoft .NET SDK bzw. Visual Studio .NET eingesetzt werden. Die Programmiersprache für den Client kann frei gewählt werden, während C# für die Beschreibung der Webservices und Proxyklassen empfohlen wird.

Folgende Schritte sind durchzuführen:

- Erstellen eines LOGON Webservice, durch die eine Autorisierung gegenüber dem Legacy Systeme ermöglicht wird.
- Erstellen eines AD (Agent Display) Webservice, durch die eine Erfolgskontrolle der Autorisierung ermöglicht wird.
- Erstellen eines TB (Terminal Information) Webservice, welcher ein Abfrageservice ist, der mehrere Transaktionsaufrufe zusammenfasst.
- Erstellen eines LOGOUT Webservice.

Das Ziel dieser Arbeit besteht darin, aufzuzeigen, dass die Businesslogik der Legacy-Mainframe-Systeme mittels vorhandener Webservice-Techniken im Internet exponiert werden kann.

- Das primäre Ziel ist die Verarbeitung von Transaktionsdaten durch ASP.NET
- Das sekundäre Ziel ist die Darstellung der Daten im Browser.

6.2 PROJEKTDURCHFÜHRUNG

In dieser Arbeit wurde eine typische Abfrage aus dem Flugbetrieb anhand einer Lufthansa-Passagierliste (NPL) praktisch abgebildet. Innerhalb der Microsoft .NET- Plattform wurde dabei C# als Programmiersprache für den Client sowie für die Beschreibung der Web-services und der Proxyklassen eingesetzt.

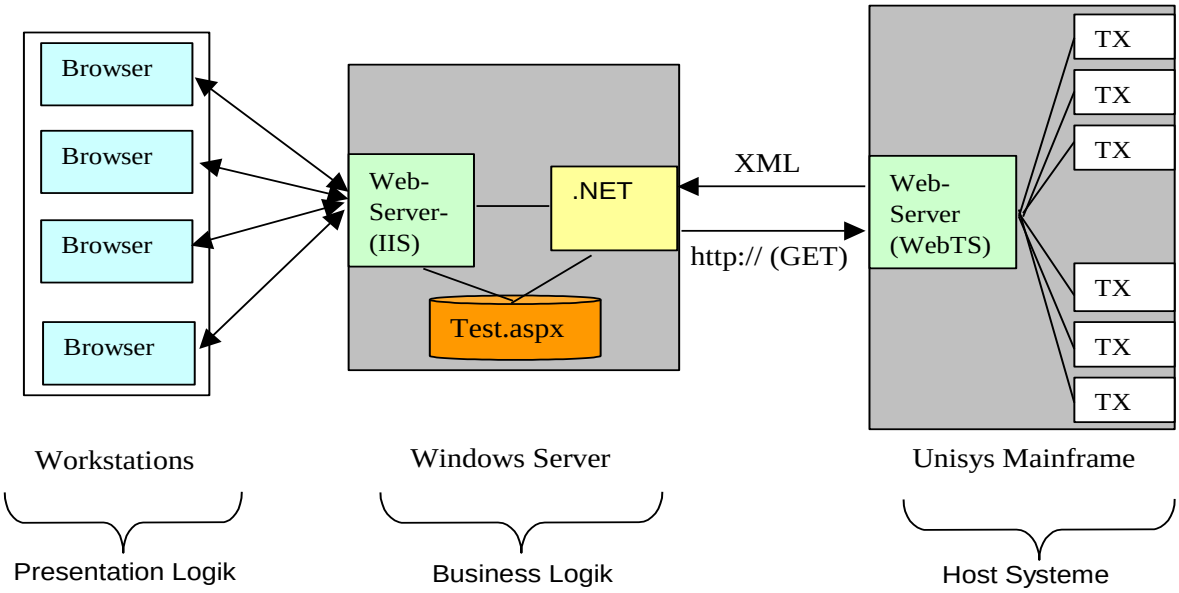


Abbildung 12 - Funktionsdiagramm der NPL-Verarbeitung

In groben Zügen funktioniert der Datenfluss folgendermaßen:

Die als Transaktionsbeispiel ausgewählte NPL besteht aus mehreren, je nach Passagier-anzahl der Linienmaschine variierenden Seiten, in denen personen- und flugbezogene Daten für den internen Gebrauch dargestellt werden. Mit Hilfe der PN-Transaktion können Seiten durchgeblättert werden.

Die verschiedensten Transaktionen der UNISYS Mainframe können jederzeit über WebTS mittels HTTP/GET- Methode aufgerufen werden.

Der Browser richtet eine Anfrage per HTTP / GET an den .NET- Webserver (in unserem Fall MS-IIS = Internet Information Server); er wiederum reicht die Anfrage an den WebTS in ihm verständlicher Weise ebenfalls mit HTTP / GET weiter.

Der Transaktionswebserver WebTS schickt daraufhin die NPL-relevanten Host-Transaktionsdaten aus der UNISYS Mainframe als XML-Response an den Windows Server. Hier im .NET werden diese XML-Daten mit CDATA unverändert gelassen, als InnerHTML-Eigenschaft einer Spanarea zur seitenweise Anzeige „gerändert“, als aspx-Datei formatiert und evtl. zwischengespeichert. Nun wird dieser Service mit WSDL als WS beschrieben und über den IIS an den Browser übermittelt.

6.3 PRAKTISCHE AUSFÜHRUNG

6.3.1 Logon

Vor der Benutzung des NPL- Systems müssen bestimmte Angaben zur Autorisierung eingegeben werden.

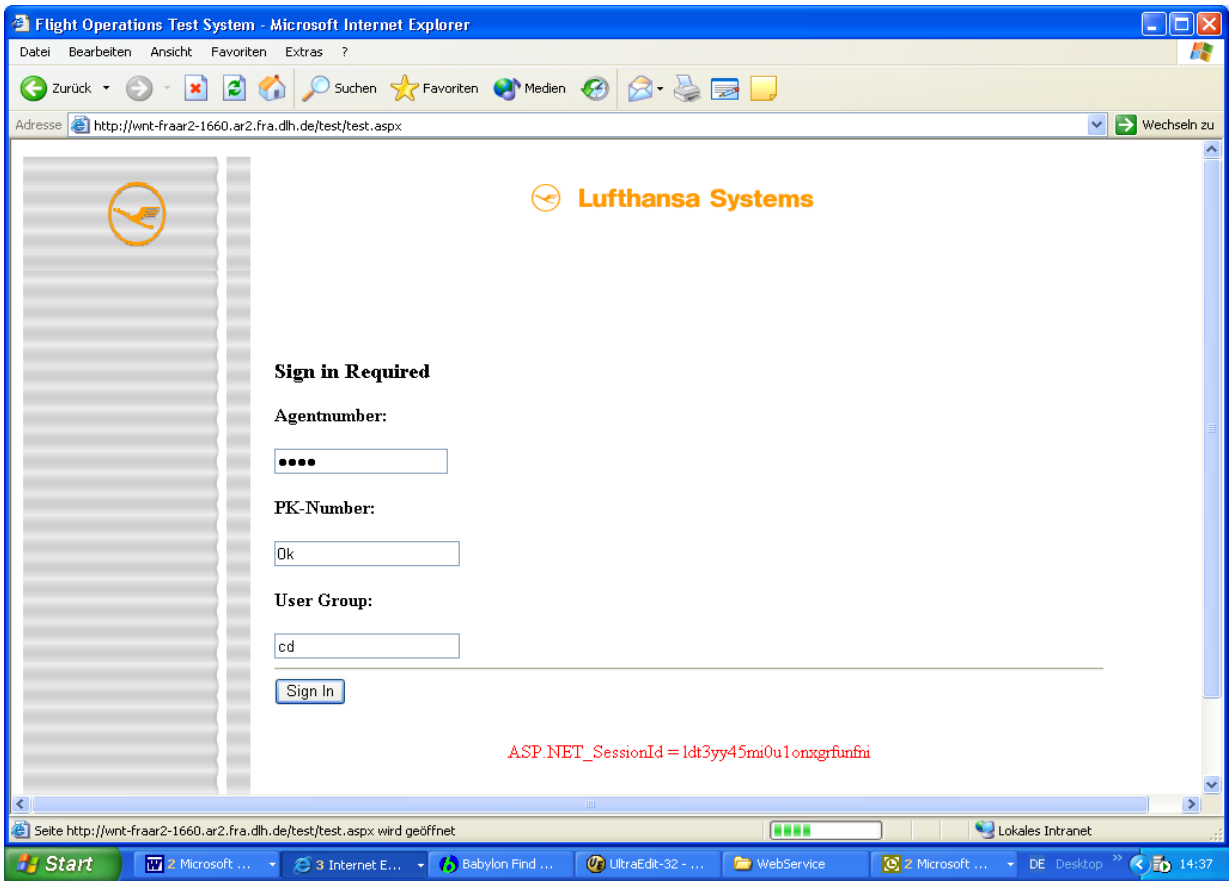


Abbildung 13 – LOGIN: Session-Autorisierung

Nach erfolgreicher Eingabe wird die Eigenschaft des Session Objekts „Sign In“ auf True gesetzt. Nach so bewilligtem Zugriff wird neue Formseite gerendert und man bekommt die TX-Menüseite. Zum Aufruf von bestimmten Transaktionen klickt man entweder die dort befindlichen Schaltflächen an oder schreibt die gewünschte Transaktion in das OFE-Eingabefeld.

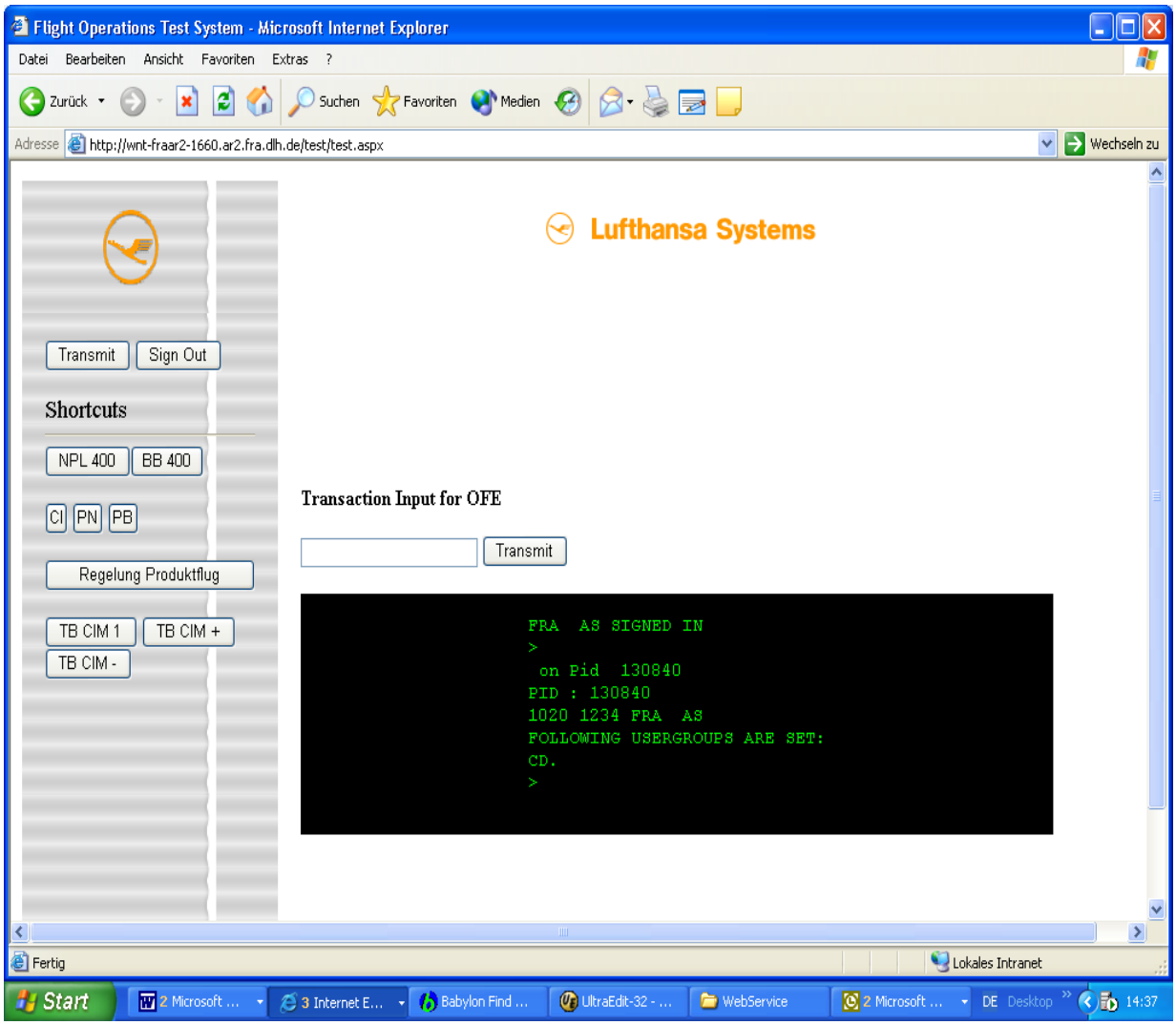


Abbildung 14- TX- Menü- Seite

Nachdem man z.B. „NPL 400“-Button direkt anklickt oder im Eingabe-Feld eingibt, kann nach dem Transmit-Befehl (eine HTTP / GET Anfrage wird abgesetzt) die NPL-Transaktion „Relevante Informationen“ abgerufen werden, die im Terminalmonitor in einem HTML Screen als Fenster dargestellt werden kann, siehe Abbildung 15.

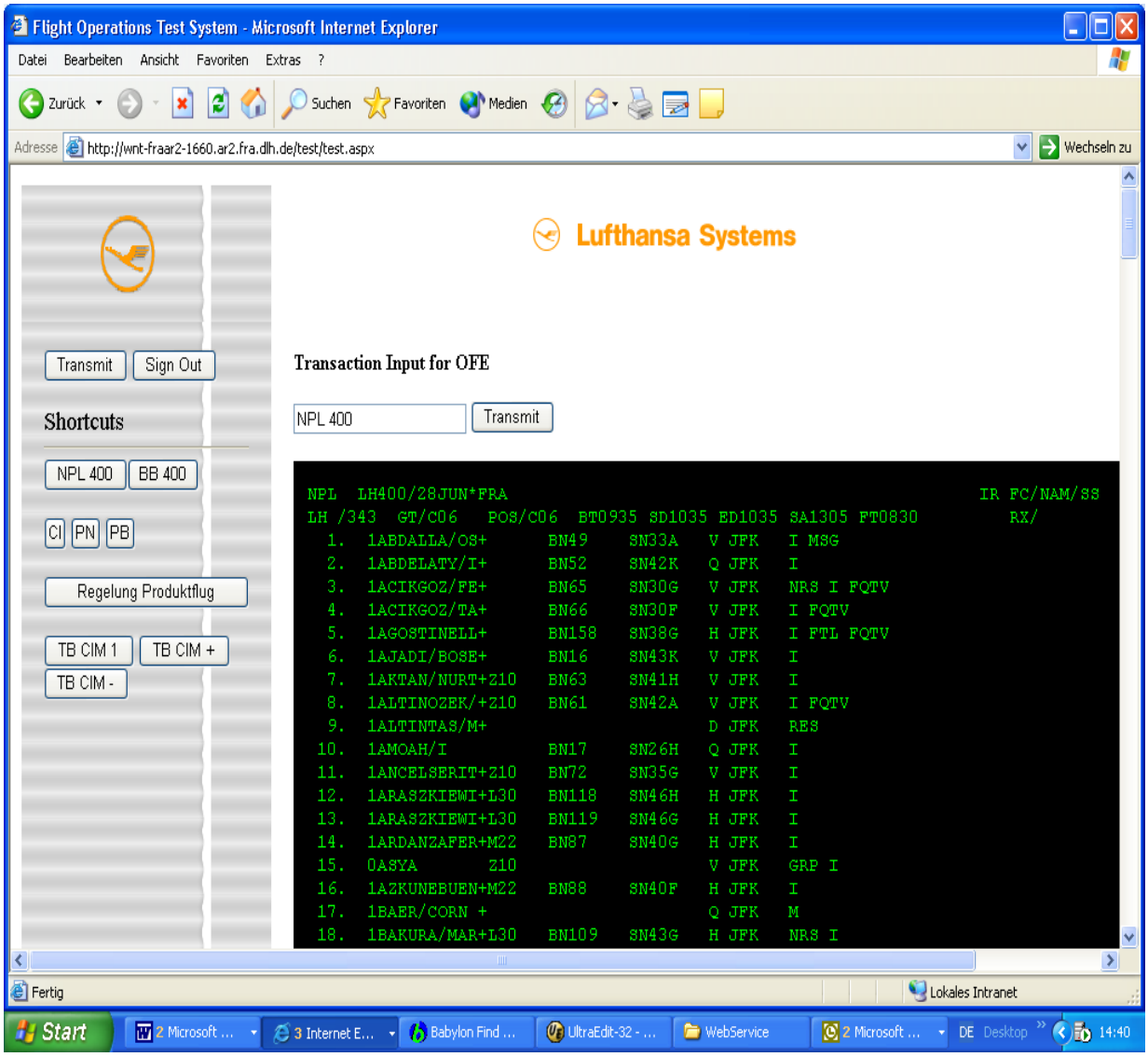


Abbildung 15 - Seitenweise Darstellung der mehrseitigen NPL 400 Transaktionen

6.3.2 Screen- Scraping mit „CDATA“

Die von WebTS ausgegebene XML- Rohdaten aus den NPL-Transaktionen werden im .NET in ein CDATA-Segment übergeben, um das „Parsen“ der Sessionsdaten durch den XML-Reader zu verhindern. Dann werden sie zur seitenmäßigen Darstellung (wie die nächsten Screen Shots) „gerändert“ und unter der aspx- Datei abgelegt, nachdem sie mit WSDL beschrieben wurden. Somit kann der .NET- Web-Server IIS auf die fertige WS zugreifen und an den Browser weiterleiten.

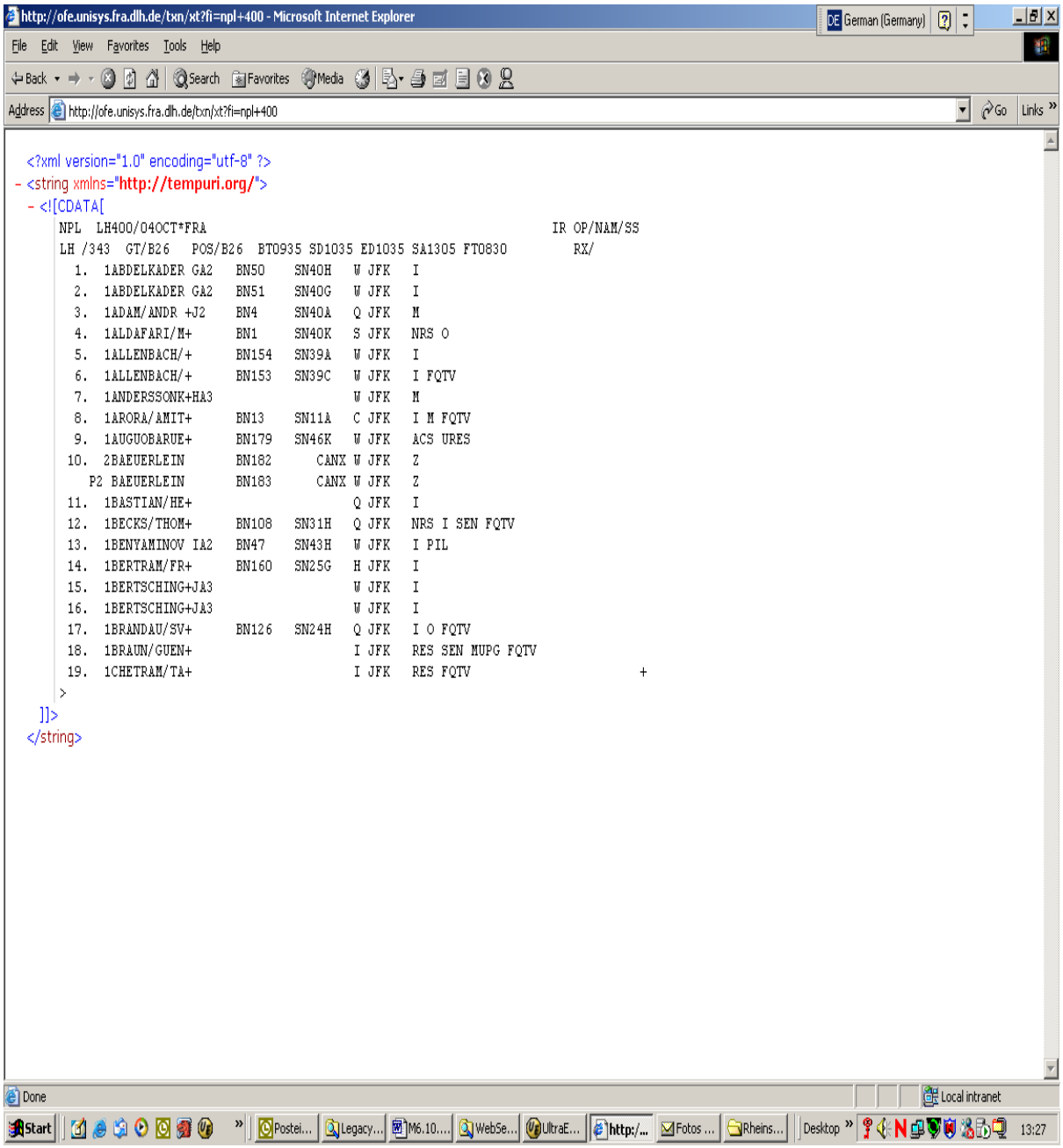


Abbildung 16- NPL Screen mit CDATA

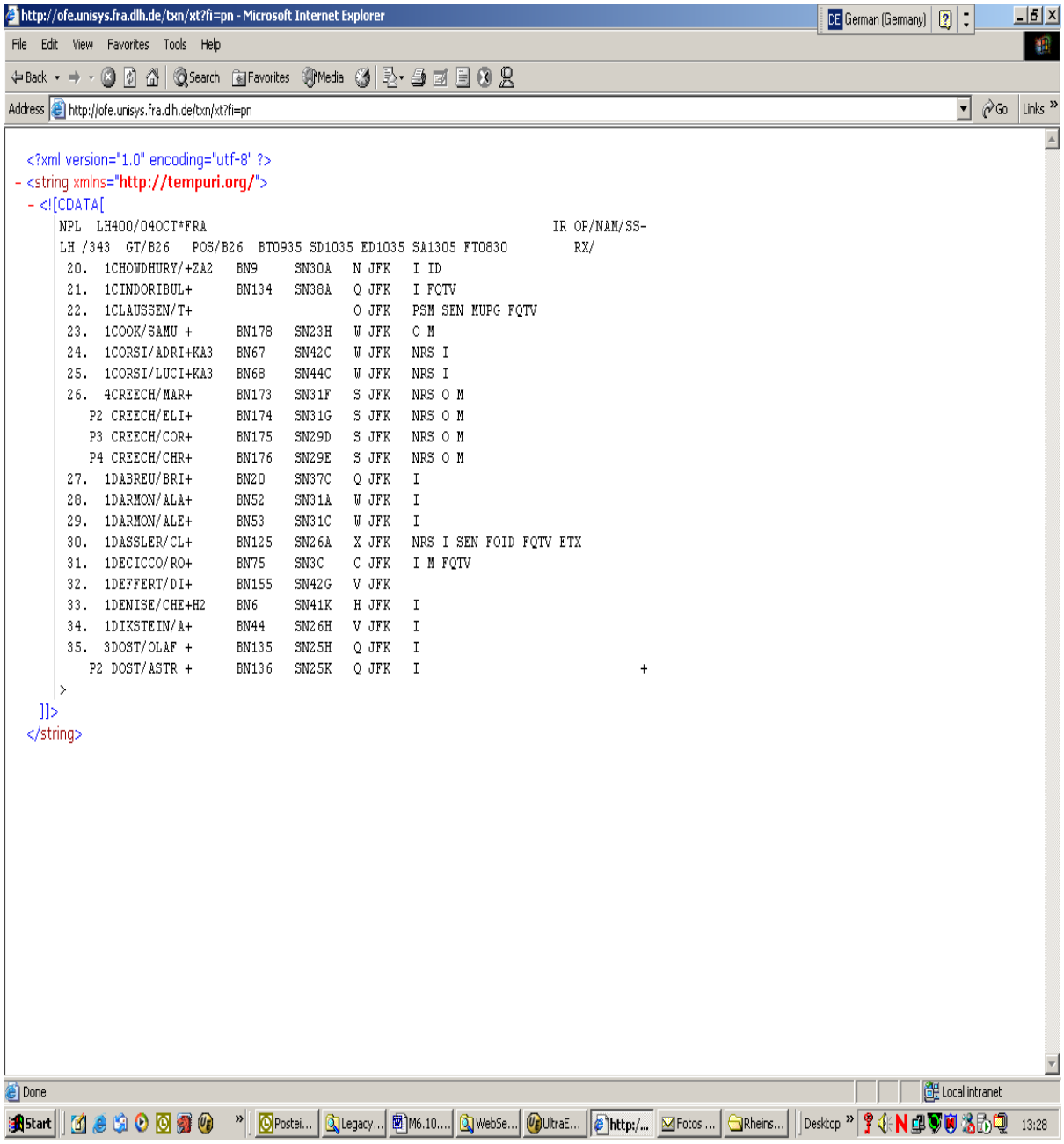


Abbildung 17- NPL Screen mit CDATA, nächste von mehreren Seiten

Implementierungen für NPL

6.3.3 Webservice Konsum für den Klient/Browser

Usas xml.cs

```
// ------------------------------------------------------------------
// This source code was auto-generated by wsdl, Version=1.0.2914.16.
// ------------------------------------------------------------------
namespace Usas {
    using System.Diagnostics;
    using System.Xml.Serialization;
    using System;
    using System.Web;
    using System.Web.Services.Protocols;
    using System.Web.Services;
    using System.Net;
    using System.IO;
    using System.Text;

public class Usas : System.Web.Services.Protocols.HttpGetClientProtocol {

        [System.Diagnostics.DebuggerStepThroughAttribute()]
        public Usas() {

  // Adresse

        this.Url = "http://ofe.unisys.fra.dlh.de/txn";
        }

        [System.Diagnostics.DebuggerStepThroughAttribute()]

[System.Web.Services.Protocols.HttpMethodAttribute(typeof(System.Web.Servic
es.Protocols.XmlReturnReader),typeof(System.Web.Services.Protocols.UrlParam
eterWriter))]
        [return: System.Xml.Serialization.XmlRootAttribute("string",
Namespace="http://tempuri.org/", IsNullable=true)]
        public string UsasTX(string fi) {
            return ((string)(this.Invoke("UsasTX", (this.Url+"/xt"), new
object[] {fi})));
        }

        [System.Diagnostics.DebuggerStepThroughAttribute()]
        public System.IAsyncResult BeginUsasTX(string fi,
System.AsyncCallback callback, object asyncState) {
            return this.BeginInvoke("UsasTX", (this.Url+"/xt"), new
object[] {fi}, callback, asyncState);
        }

        [System.Diagnostics.DebuggerStepThroughAttribute()]
        public string EndUsasTX(System.IAsyncResult asyncResult) {
            return ((string)(this.EndInvoke(asyncResult)));
        }

        [System.Diagnostics.DebuggerStepThroughAttribute()]
      [System.Web.Services.Protocols.HttpMethodAttribute(typeof(System.Web.
Services.Protocols.TextReturnReader),typeof(System.Web.Services.Protocols.
UrlParameterWriter))]
```

```
public Mpo UsasHtmlTX(string fi) {
            return ((Mpo)(this.Invoke("UsasHtmlTX", (this.Url+"/ut"), new
object[] {fi})));

        }

        [System.Diagnostics.DebuggerStepThroughAttribute()]
        public System.IAsyncResult BeginUsasHtmlTX(string fi,
System.AsyncCallback callback, object asyncState) {
            return this.BeginInvoke("UsasHtmlTX", (this.Url+"/ut"), new
object[] {fi}, callback, asyncState);
        }

        [System.Diagnostics.DebuggerStepThroughAttribute()]
        public string EndUsasHtmlTX(System.IAsyncResult asyncResult) {
            return ((string)(this.EndInvoke(asyncResult)));
        }

    }

public class Mpo {

System.Web.Services.Protocols.MatchAttribute("<TEXTAREA.*?>(.*?)</TEXTAREA"
)]
        public string MpoText;


[System.Web.Services.Protocols.MatchAttribute("<applet(.*?)</APPLET>")]
        public string Dps;

        [System.Web.Services.Protocols.MatchAttribute("PID:(.*?)\r")]
        public string AgentPid;



   }
}
```

Regular Expression

6.3.4 Browser-Layout

Test.aspx

```
<%@ Page Language="C#" Debug="false" %>

<%@ Import Namespace="Usas" %>
<%@ Import Namespace="System.Web.Services" %>
<%@ Import Namespace="System.Web.Services.Protocols" %>
<%@ Import Namespace="System.Net" %>
<%@ Import Namespace="System" %>
<%@ Import Namespace="System.Xml.Serialization" %>
<%@ Import Namespace="System.Diagnostics" %>

<html>
<head>
<title>Flight Operations Test System</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">

<script language="C#" runat="server">
int tbp;
string rsp, msg;
string npl = "NPL 400";
string bb = "BB 440";
string ci = "CI";
string pn = "PN";
string pb = "PB";
string prod = "CI PAD/PROFLUG1";
string tb = "TB CIM";
int loop;
string myKey, myValue;
protected void Page_Load(Object Sender, EventArgs E)
   {
      if (!Page.IsPostBack) {
                  tbp = 1;
                  Output.InnerHtml = ""+Session["msg"];
                  Shortcut.Text = "Shortcuts" ;
                  }
               else
                  try {
                        tbp = int.Parse((string)Session["TB"]);
                  } catch (Exception ex) { }

            Session["TB"] = tbp.ToString("d2");
                  string msg = "";
            if (Session["registered"]==null) {
                  Register myReg = new Register();
                  try {
       myReg.RegisterService("WSPID="+130902+"&WSIDENT=test");
                        Session["registered"] = true;
                        HttpCookieCollectionmyCookies= Request.Cookies;
                        for (loop=0; loop<myCookies.Count; loop++) {
                              myKey = myCookies.GetKey(loop);
                              myValue = myCookies[loop].Value;
                              msg += "\r"+myKey+" = "+myValue;
                              }
                  }
                 catch (Exception ex) { Message.InnerHtml = ex.Message; }
            }
            Message.InnerHtml = msg;
   }
```

```
// Für Aufruf von NPL 400

public void Submit_NPL (Object Sender, EventArgs E)
   {
      myTx.Value = npl;
      Usas ut = new Usas();
      Message.InnerHtml = "";



try {                                                    While Loop, für nächste seite
         rsp = ut.UsasTX(npl);
         msg = rsp;
         while (rsp.IndexOf("+\r")>0) { rsp= ut.UsasTX("pn"); msg += rsp;}
        }
    catch (Exception e) {
     try {

        Mpo mrsp = ut.UsasHtmlTX(myTx.Value);
        if (mrsp.Dps!=null) {
        msg = "<APPLET"+mrsp.Dps+"</APPLET>";
                              }
        if (mrsp.MpoText!=null) {
                 msg += mrsp.MpoText;
                                  }
        if (mrsp.AgentPid!=null) {
                 msg += "<p>Found PID "+mrsp.AgentPid;
                                   }
         }
      catch (Exception ex) { Message.InnerHtml = ex.Message; }
                  }
Output.InnerHtml="<pre>"+msg+"</pre><p>UsingPid"+Session["Pid"]+"</body>";
   }

// Für Aufruf von CI Transaktion

public void Submit_CI (Object Sender, EventArgs E)
   {
      myTx.Value = ci;
      Usas ut = new Usas();
      Message.InnerHtml = "";
      try
      {
         rsp = ut.UsasTX(ci);
         msg = rsp;
         while (rsp.IndexOf("+\r")>0) { rsp= ut.UsasTX("pn"); msg += rsp;}
        }
      catch (Exception e) {
      try {
         Mpo mrsp = ut.UsasHtmlTX(myTx.Value);
         if (mrsp.Dps!=null) {
         msg = "<APPLET"+mrsp.Dps+"</APPLET>";
                             }
         if (mrsp.MpoText!=null) {
                   msg += mrsp.MpoText;
                                 }
if (mrsp.AgentPid!=null) {
        msg += "<p>Found PID "+mrsp.AgentPid;
                          }
                        }
catch (Exception ex) { Message.InnerHtml = ex.Message; }
                        }
Output.InnerHtml="<pre>"+msg+"</pre><p>UsingPid "+Session["Pid"]+"</body>";
   }
```

// Die oben beschriebenen Implementierungsteile für NPL und CI sind
// repräsentativ für alle Transaktionen auf der Menüseite. Für die
// restlichen Transaktionen müssten die Implementierungen wiederholt werden.

// Nachfolgend Implementierung für Login und Logout

```
public void Submit_SI (Object Sender, EventArgs E)
   {
      int myPidStart, myPidEnd;
      string myPid;

      Usas ut = new Usas();
      UsasRes utres = new UsasRes();
      Message.InnerHtml = "";

      try
      {
Mpomsg=ut.UsasHtmlTX("SI"+mySignIn.Value+"/"+myPK.Value+"//"+myUG.Value);
         Session["Agent"] = mySignIn.Value;
         Session["PK"] = myPK.Value;
         Session["UG"] = myUG.Value;
         if (msg.MpoText.IndexOf("?") < 0) {
            Session["signin"] = true;
            Mpo rsp = ut.UsasHtmlTX("AD");
            myPidStart = rsp.MpoText.IndexOf("PID :");
            myPidEnd = rsp.MpoText.IndexOf("\r",myPidStart);
            myPid=rsp.MpoText.Substring(myPidStart+5,(myPidEnd-myPidStart-
5+1));
            Session["Pid"] = myPid;
            string cmsg = "";
            ut.UsasHtmlTX("tst acb sys");
            Message.InnerHtml = cmsg;
            Output.InnerHtml="<pre>"+msg.MpoText+"\r
onPid"+myPid+rsp.MpoText+"</pre>";
         } else {
            Output.InnerHtml = "<pre>"+msg.MpoText+"</pre>";
         }

        }
      catch (Exception e) { Message.InnerHtml = e.Message; }
    }

</script>
</head>

<body bgcolor="#FFFFFF" text="#000000">
<tablewidth="1003"border="0"cellpadding="20"cellspacing="0"
mm:layoutgroup="true">
  <tr>
    <tdbackground="./images/back.gif"width="138"height="79"
valign="top"><img src="Images/headneu3.gif" width="150" height="60"></td>
    <tdwidth="469" valign="top"><img src="Images/LHsystems.gif" width="254"
height="32" align="right"></td>
    <td></td>
    <td></td>
  </tr>
  <tr>
<td height="525" valign="top" background="./images/back.gif">
<form method="POST" runat="server">
<% if (Session["signin"] != null) { %>
<asp:button text="Transmit" OnClick="Submit_TX" runat="server"/>
<asp:button text="Sign Out" OnClick="Submit_So" runat="server"/><p>
```

```
<asp:Labelid=Shortcutfont-name="Verdane"font-size="14pt"
runat="server"/><hr>

<asp:buttontext="NPL400"OnClick="Submit_NPL" runat="server"/>
<asp:buttontext="BB440"OnClick="Submit_BB" runat="server"/><p>
<asp:button text="CI" OnClick="Submit_CI" runat="server"/>
<asp:button text="PN" OnClick="Submit_PN" runat="server"/>
<asp:buttontext="PB"OnClick="Submit_PB" runat="server"/><p>
<asp:buttontext="RegelungProduktflug"OnClick="Submit_PROD"runat="server"/><
p>
<asp:button text="TB CIM 1" OnClick="Submit_TBS" runat="server"/>
<asp:button text="TB CIM +" OnClick="Submit_TBF" runat="server"/>
<asp:button text="TB CIM -" OnClick="Submit_TBB" runat="server"/>
<% } %>
</td>

<td valign="center" width="865">
    <% if (Session["signin"] == null) { %>
      <h3>Sign in Required</h3>
      <h4>Agentnumber:</h4>
<inputtype=passwordid="mySignIn" runat="server"/><p>
      <h4>PK-Number:</h4><input type=text id="myPK" runat="server"/><p>
      <h4>User Group:</h4><input type=text id="myUG" runat="server"/><hr>
 <inputtype=submitvalue="SignIn" OnServerClick="Submit_SI" runat="server"/>
      <% } else { %>
      <h4>Transaction Input for OFE</h4>
      <Input type=text id="myTx" runat="server"/>
      <asp:button text="Transmit"  OnClick="Submit_TX" runat="server"/>
      <% } %>
</form>

<% if (Session["signin"] == null) { %>
<div align="center" style="background-color: #FFFFFF"><% } else { %>
<div align="center" style="background-color: #000000"><% } %>
<table cellpadding="10" border="0">
     <tr>
        <td>
    <span id="Message" style="color: #FF0000" runat="server"/>
      </tr></td>
     <tr>
<% if (Session["Pid"] != null) { %>
        <td style="background-color: #000000">
<spanid="Output"style="font:8ptcourier;color:#00FF00;background-color:
#000000" runat="server"/>


</td>
         <% } %>
      </tr>
    </table>
</div>
    </form></td>
  </tr>
</table>
</body>
</html>
```

6.3.5 Service Beschreibung

Usas tx.wsdl

```
<?xml version="1.0" encoding="utf-8"?>

<definitionsxmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:s0="http://tempuri.org/"targetNamespace="http://tempuri.org/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <s:schemaattributeFormDefault="qualified"elementFormDefault="qualified"
targetNamespace="http://tempuri.org/">
      <s:element name="UsasTX">
        <s:complexType />
      </s:element>
      <s:element name="UsasTXResponse">
        <s:complexType>
          <s:sequence>
            <s:element   minOccurs="1"   maxOccurs="1"   name="UsasTXResult"
nillable="true" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="string" nillable="true" type="s:string" />
    </s:schema>
  </types>
<message name="UsasTXHttpGetIn">
    <part name="fi" type="s:string" />
  </message>
  <message name="UsasTXHttpGetOut">
    <part name="Body" element="s0:string" />
  </message>
</portType>
  <portType name="UsasHttpGet">
    <operation name="UsasTX">
      <input message="s0:UsasTXHttpGetIn" />
      <output message="s0:UsasTXHttpGetOut" />
    </operation>
  </portType>
  <binding name="UsasHttpGet" type="s0:UsasHttpGet">
    <http:binding verb="GET" />
    <operation name="UsasTX">
      <http:operation location="/UT" />
      <input>
        <http:urlEncoded />
      </input>
      <output>
        <mime:mimeXml part="Body" />
      </output>
    </operation>
  </binding>
    <service name="UsasTX">
    <port name="UsasHttpGet" binding="s0:UsasHttpGet">
      <http:address location="http://ofe.unisys.fra.dlh.de/txn" />
    </port>
    </service>
</definitions>
```

6.4 PROJEKTANALYSE/- BEWERTUNG

Wie diese Arbeit gezeigt hat, können die Transaktionsdaten sehr wohl durch ASP.NET verarbeitet und als Webservices dargeboten werden. Der UNISYS-WebTS ermöglicht die Nutzung von Transaktionen durch das Internet mittels eines Browsers mit einem HTTP/ GET- Protokoll formatierten URL. Dabei kommt uns sehr gelegen, dass er die Sessionsdaten als XML–Response ausgibt. So bleibt der Aufwand für die nachfolgende Implementierung gering.

Bei dieser ASP.NET basierenden Anwendung wurde innerhalb einer Standalone - Inbetriebnahme eine Auswahl von Transaktionen autorisiert und mit Hilfe eines „gestutzten“ Webservices ausgeführt. Gestutzt deshalb, weil die Beschreibung mit WSDL alleine nicht ausreicht, um sie als autonome Webservices über aktive Webpages zu verkaufen. Dazu müssten die NPL-Daten im CDATA-Segment durch den XML-Reader geparst, mittels XSL definiert und last but not least mit XSLT zur Darstellung im Browser transformiert werden.

Die sicherheitsrelevanten Kriterien sind wegen dem Standalone-Einsatzgebiet nur in Ansätzen implementiert, lediglich die Autorisierung wurde gegenüber der UNISYS-Mainframe durchgeführt.

Der Prototyp dieser Arbeit konnte nur im gewissen Rahmen erfolgreich betrieben werden. Für einen kommerziellen Einsatz sind außer den weiter oben angegebenen Schritten noch zusätzliche Entwicklungsstufen nötig. Die Implementierung von Transaktionen mit Webservices wirft dabei viele Fragen auf:

- Sicherheitsrelevante Daten sollten nur vom autorisierten Personenkreis über den Web bezogen werden.
- Die Transaktionsdaten sollten kategorisch und eindeutig den einzelnen Benutzer-profilen zugeordnet werden können.
- Sicherheitsfunktionen und Zugriffskontrollen müssen innerhalb wie außerhalb der Unternehmensgrenzen strikt erfüllt werden.
- Komplexe Geschäftsprozesse mit mehreren Teilnehmern erfordern Transaktions-unterstützung und Orchestrierung von Webservices.

In sämtlichen Punkten besteht Konsensbedarf für universelle Standards. In verschiedenen Foren werden noch ungelöste Probleme rund ums Webservice heftig diskutiert, wie z.B. BPEL für Transaktionen.

Erst wenn die Sicherheit und die Transaktionsunterstützung geregelt sind, erscheinen wirklich universelle und global operierende UDDIs in greifbarer Nähe.

Zumindest können jetzt weitverzweigte Betriebe oder Unternehmenszusammenschlüsse sowie interessengekoppelte Bündnisse (STAR-ALLIANZ z.B.) durch Errichtung eines gemeinsamen, UDDI basierten Service-Portals ihre Interessen untereinander koordinieren und erst wenn die oben umrissenen Mängeleigenschaften der WS beseitigt werden, mit anderen Unternehmen oder Allianzen über deren Portale kommunizieren.

7. SCHLUSSBETRACHTUNG

7.1 RISIKEN- UND KOSTENANALYSE

Schnelle Veränderungen von Marktbedingungen erfordern rasche Anpassung durch Flexibilität und Schnelligkeit. Die Sicherung und der Ausbau der vorhandenen Geschäftsfelder ist dabei erklärtes Ziel, ebenso die Kosteneinsparung durch Verbesserung der Geschäftsprozesse. Den Kampf um Marktanteile und Kostendruck muss jeder individuell begegnen; z.B. müssen Industrieunternehmer die Partner und Zulieferer leicht und flexibel in ihre Strategien einbinden können, während beim Handel die Erhöhung der Schnelligkeit und Flexibilität sowie die Verbesserung der Anpassungsfähigkeit gefragt ist. Der Dienstleistungssektor muss darüber hinaus aktuelle Position sichern und Geschäftsfelder ausbauen.
Vgl.[CapGem]

Die Systemkonfiguration vieler Unternehmen mit den vielen Individuallösungen und Standard- ERP- Systemen, E- Business-, Data Warehouse- und CRM- Anwendungen sind wegen ihrer Heterogenität kaum noch ganzheitlich zu beherrschen und ihre mangelnde Integrationsvermögen gefährdet nicht zuletzt die Wirtschaftlichkeit der IT- Projekte.
Vgl.[SoftAG]

Viele Unternehmen mussten leidvoll erfahren, wie zeitaufwendig, teuer und risikoreich der Einsatz von neu entwickelter Software bei ihren neuen Geschäftskonzepten ist. Der **R**eturn **o**n **E**xisting **I**nvestment wirft in Zeiten knapper IT- Budgets die entscheidende Frage auf, welche Rolle die Legacy- Systeme in ihren Bemühungen spielen. Sie müssen am Anfang einer kostenbewussten Modernisierungsstrategie entscheiden, welche Altanwendungen für neue Geschäftsmodelle und insbesondere für die neuen Projekte wie CRM Vorrang haben. Dabei können sie aus den Investitionen und Weiterentwicklungen der Anwendungen vergangener Jahre auch in diesem Zeitalter rasanter Entwicklungen noch ROEI- gehörigen Kapital schlagen, wenn sie intelligente Integrationskonzepte mit einem überschaubaren Aufwand an Zeit und Kosten einsetzen. Vgl.[Bkm02]

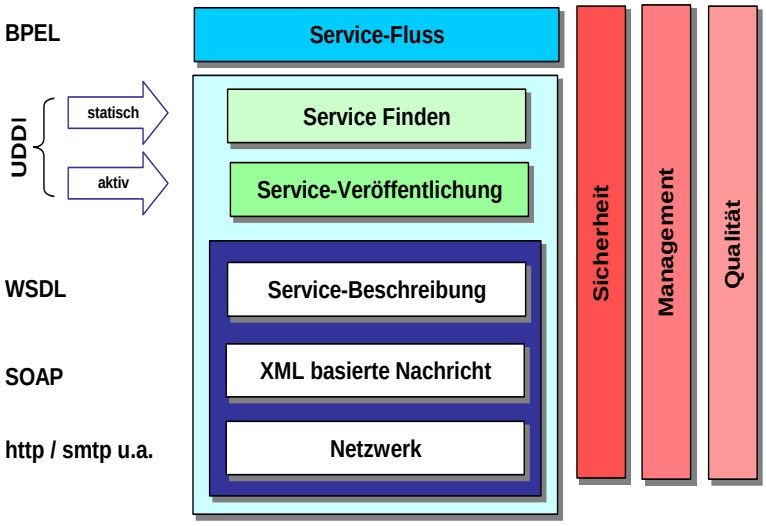
Obwohl die heute verfügbaren EAI- Anwendungspakete mit vielfältigen und aufeinander abgestimmten Tools an einigen Projekten erfolgreich eingesetzt werden, hält sich das Interesse vieler potentieller Anwender in Grenzen. Zum einen dürfte es daran liegen, dass diese Produkte als proprietäre Middleware mit komplexen Tools beträchtliche Anlern- und Implementierungsaufwand „kosten“. Häufig werden viele Funktionen nicht gebraucht, die im Paket mitbezahlt wurden oder der Entwickler benötigt trotz der Vielzahl von Tools noch weitere, die er dazukaufen müsste. Zum anderen steht die Vermarktung von rudimentären

Tools zur Geschäftsprozess- Integration bei den EAI- Herstellern im Vordergrund, obwohl die Anforderungen vieler Integrationsprojekte auf der Daten- und Programmebene liegen. Nicht zuletzt sind diese Pakete teuer, was die Projektkosten explodieren lässt. Deshalb sind bei mittelständischen Unternehmen und auch bei abteilungsbezogenen Projekten billigere und einfacher realisierbare Produkte gefragt: Message- orientierte Middleware etwa kommt wesentlich günstiger als ein umfangreiches EAI- Paket und neuerdings gibt es Tools mit der Datenbeschreibungssprache XML als Open Source sogar kostenlos. Vgl.[News02] Diese unabhängigen Werkzeuge lassen sich flexibel und kostengünstig einsetzen. Dabei eröffnet das Konzept von Webservices durch die damit mögliche hohe Daten- und Prozess-Integration ganz neue Möglichkeiten und unternehmensübergreifende Chancen, ohne dass man dabei in der Vergangenheit getätigte Investitionen verlieren muss.

Noch investieren Unternehmer nur zögernd und abwartend in Webservice, aber laut einer aktuellen Studie von Forrester Research werden ihre Ausgaben für die Implementierung von strategischen WS in nächster Zeit drastisch steigen (müssen). Trotz der hohen Erwartungen in WS sind sich die Führungskräfte deren Grenzen bewusst. Vielen ist die Sicherheit bei der Benutzung von WS eine bislang ungelöste Lücke, wie einfach ist doch die Möglichkeit, unternehmensspezifische Backend- Applikationen über XML- Schnittstellen öffentlich ins Web zu tragen. Auch wenn die WS die Wertschöpfung aus früheren Software- Investitionen ermöglicht, warnt Forrester dabei vor zu großer Eupherie: Obwohl die Lizenzierung eines sehr teueren Integrationsservers entfalle, seien die Implementierungskosten doch beachtlich. Die wahren Kosten würden auf Grund von Consulting- und Arbeitszeitkosten viel höher liegen als in der Startphase angenommen. In absehbarer Zeit würden WS bei allen IT-Projekten Einzug halten. Obwohl die Integrationskosten erheblich sinken werden, wären dabei die Projektkosten weiterhin hoch, während IT- Verantwortliche noch mit der Gestaltung von Unternehmensprozessen und zeitlicher Optimierung zu kämpfen hätten. Vgl.[golem02]

Nichtsdestotrotz rechnet sich der Einsatz von WS, wie viele Unternehmen es schon erfolgreich vorgemacht haben. Z.B. konnte die Swiss Interbank Clearing AG in relativ kurzer Zeit von einer EDIFACT-Infrastruktur zu einer Webservice-Integrationsplattform zuverlässig, effizient, sicher und kostengünstig überführt werden. Vgl.[SwIntbk]

7.2 ZUSAMMENFASSUNG UND AUSBLICK



Webservices sind nunmehr ein fester Bestandteil von Integrationsbemühungen geworden, um nicht nur innerhalb des Unternehmens Altlasten aufzuräumen, sondern mittlerweile auch zwischen den Partnern, Kunden und Lieferanten zu koordinieren.

Sie können mit ihrer Schlichtheit und Implementierungseleganz bisherige Integrationstools wie CORBA, DCOM u.a. alt aussehen lassen.

Abbildung 18 - Webservices Stack, Vgl.[ibmstack]

Ihre textbasierten Bestandteile SOAP,WSDL und UDDI arbeiten mit XML und HTTP, die das enorme Potenzial des Internet erst jetzt richtig zur Geltung bringen können.

Diese relativ einfache Konstruktion der WS könnte ihnen aber zum Verhängnis werden, wenn die noch fehlenden bzw. unzureichenden Funktionen wie Sicherheit, Transaktionssteuerung oder Workflowmechanismen nicht alsbald ergänzt würden. Hierbei steckt noch jede Menge Entwicklungsbedarf, der sich in verschiedensten intensiven Diskussionen und Vorschlägen resultiert.

Dabei ist immens wichtig, sich wie bei XML auf gemeinsame Standards zu einigen. Mehrere proprietäre Standards könnten sonst den vorteilhaften Mehrwert von WS für die Integration entscheidend schmälern.

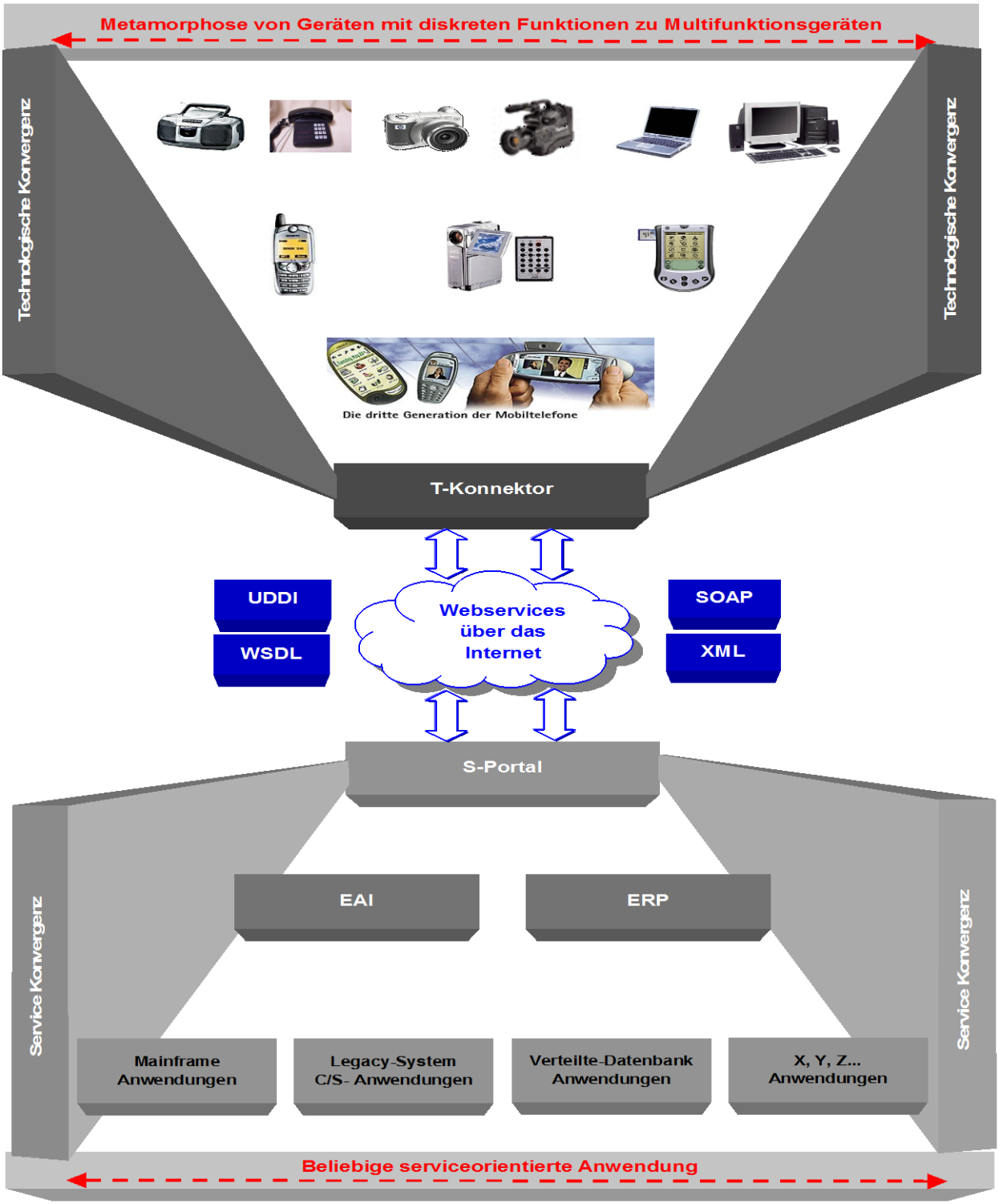


Abbildung 19 - Technologie und Service Konvergenz

Die technologischen Innovationen lassen die Geräte immer weiter schrumpfen, wobei gleichzeitig die Multifunktionalitäten zunehmen, bis sie in den Technologie-Konnektor übergehen. Die ständig wandelnden und wachsenden Geschäftsanforderungen zwingen die verschiedenen Service-Anwendungen zur Integration, bis hin zum Service-Portal.

In nächster Zukunft werden die Webservices als Bindeglied zwischen den beiden Konvergenzen eine überaus wichtige Rolle spielen..... Zum Wohle aller !!!

Glossar

ACID-Prinzip	Atomicity, Consistency, Integrity, Durability; unteilbare Folge von Anweisungen, durch die ein System von einem Konsistenten Zustand in einen anderen überführt wird
API	Application Programming Interface; standardisierte Schnittstelle zur Programmierung
Applet	Java-Programm, das einen Browser als virtuelle Maschine nutzt
BPEL	Business Process Execution Language for Web Services; von den IBM, Microsoft u.a. definierte Sprache zur Beschreibung von Geschäftsprozessen. Sie basiert auf XML und definiert, wie unterschiedliche Geschäftsprozesse zusammenwirken.
B2B	business to business; elektronische Geschäftsabwicklung zwischen Unternehmen
B2C	business to customer; elektronische Geschäftsabwicklung zwischen Unternehmen und Privaten
CORBA	Common Open Request Broker Architektur; standard für die netztransparente Kommunikation zwischen verteilten Objekten
CRM	Customer Relationship Management; verwaltung der Kundenbeziehungen. CRM-Systeme sammeln und analysieren Informationen über den Kunden und das Kundenverhalten. Ziel ist, die Kunden optimal und individuell betreuen zu können
EAI	Enterprise Application IntegrationTechnologien; welche automatisiert die Kommunikation und Interoperabilität zwischen unterschiedlichen Anwendungen und Geschäftsprozessen innerhalb undzwischen Organisationen ermöglichen
GUI	Graphical User Interface; grafische Benutzeroberfläche von Browser
HTML	Hyper Text Markup Language; die Sprache zur Layout-Beschreibung von Dokumenten
HTTP	Hyper Text Transfer Protocol; protokoll zur Kommunikation zwischen Klient (Browser) und Webserver Mainframe Großrechner
IIS	Internet Information Server; webserver von Microsoft
RMI	Remote Methode Invocation; mechanismus zum Aufruf von Methoden anderer Java-Objekte im Netz
ROI	Statisches Verfahren der Investitionsrechnung. Man definiert als Verhältnis des Jahresüberschusses einer Investition zum Kapitaleinsatz, als Maß für die Vorteilhaftigkeit einer Investition angesehen

RPC	Remote Procedure Call; mechanismus zum Aufruf von Funktionen im Netz verfügbarer Server
SMTP	SimpleMailTransferProtocol; standardprotokoll zum Versand von E-Mails
TCP/IP	Transmition Control Protocol / Internet Protocol; protokollsuite für die verbindungsorientierte und verbindungslose Übertragung von Datenpaketen und Datenströmen
Terminal	Bildschirm mit angeschlossener Tastatur, reine Ein- und Ausgabestelle; bei LH-Systems z.B. für den Zugang zu den Unisys-Mainframes verwendet
Transaction	Transaktion folgt dem ACID-Prinzip; kann mit unterschiedlichen Parameter gestartet werden
URL	Uniform Ressource Locator; eindeutige Beschreibung einer Ressource im Intra-/Internet
Webserver	Software zur Verwaltung und Bereitstellung von Übertragungsdiensten wie FTP und http, meist auf Basis von TCP/IP
WebTS	Web Transaction Server von der Firma UNISYS
WSFL	Webservices Flow Language; von der Firma IBM ; beschreibt den Ablauf von Geschäftsprozessen innerhalb von Webservices.
XLANG	*Web Services for Business Process Design von der Firma Microsoft;* beschreibt den Ablauf von Geschäftsprozessen innerhalb von Webservices.
XSLT	*eXtensibleStylesheet* Language Transformations; Programmiersprache für die Transformation von XML-Dokumenten

Abbildungsverzeichnis

Literaturverzeichnis

[Bkm02] Nils Brauckmann: *Bewährtes für die Zukunft - Legacy goes Future*, 06/2002
http://www.contentmanager.de/magazin/artikel_188_legacy_systeme ebusiness.html

[CapGem] Cap Gemini: *Der Markt für Webservices,* Juni 2002
http://www.de.cgey.com/servlet/PB/show/1004620/Web-Services.pdf

[Colan01] Colan, M.: SOAP*: Security and Reliability, Issues and Solutions*.
ftp://www6.software.ibm.com/software/developer/library/mcolan/SOAP-Security-Issues-Solutions.pdf
IBM developerWorks, 2001.

[Core02] *XML Encryption Syntax and Processing,* 03.10.2002
W3C Proposed Recommendation
http://www.w3.org/TR/2002/PR-xmlenc-core-20021003/

[c't 02] C'T Magazin für Computertechnik: *Web Services lernen Transaktionen*,
heft 18, seite 44,2002

[Decr02] W3C Proposed Recommendation: *Decryption Transform for XML Signature*
03 October 2002
http://www.w3.org/TR/2002/PR-xmlenc-decrypt-20021003

[Ecinweb] Tobias Arndt: *Webservices wollen Web erobern*,11/2001
http://www.ecin.de/technik/webservices/

[Eich02] F.Eichhorn & Muhammad F.Kaleem: *Integration ohne Grenzen*;
XML Magazin & Web Services ,Ausgabe 2/02

[EichSec] Felix Eichhorn, Diplomarbeit: *Evaluation von Webservice-Techniken für den Einsatz zur Business-to-Business Integration (B2BI), 04/02*
http://www.ti5.tu-harburg.de/Publication/2002/thesis/eichhorn02/eichhorn02.pdf

[golem02] Golem-Studie: *Web Services teurer als von Anbietern behauptet*, 06/02
http://www.golem.de/0206/20535.html

[**Gruhn02]** Gruhn V., Schöpe L.: *Integration von Legacy-Systemen*
http://ls10-www.informatik.uni-dortmund.de/~schoepe/artikel/memo124.pdf
Universität Dortmund, Informatik Centrum Dortmund e.V

[Gümüs] Gümüs O., Schiefer A.: E-Commerce Architektur*: Einbindung von Legacy Systemen*
http://www.ifs.univie.ac.at/ec2001/05/EC-SEM_final.doc

[HoffR] Dr.Ronald Hoffmann : *Was bedeutet EAI ?*
http://www.software-marktplatz.de/berichte/EAI_FR.PDF

[ibmfkt] IBM 06/02
http://www-5.ibm.com/de/software/enews/essay/2002-06-06-ess-1.html

[ibmstack] Judith M. Myerson: *Advancing the Web services stack*, 06/02
http://www-106.ibm.com/developerworks/library/ws-wsa/
IBM developerworks

[iicm01] iicm, SOAP
http://www.iicm.edu/research/seminars/ws_00/pacnik_strohmaier.pdf

[Jacada] Jacada, White Paper: Integration von Legacy-Systemen: *Chancen für E-Business-, CRM- und ERP-Umgebungen*
http://www.jacada.de/de/Produkte/Legacy_Systeme_Integration.htm

[Jeckle] Jeckle : *Webservice Definition*
http://www.jeckle.de/webServices/index.html#directories

[Jeckle02] Jeckle : *Webservice Architekturen*
http://www.jeckle.de/files/wsArch.pdf

[MS.Net] Microsoft .NET Crashkurs;Vasters, Oellers, javidi, Jung, Freiberger, DePetrolli
Microsoft Press 2001

[News02] Informationsweek IT Magazin :*Integration mit universellen Werkzeugen*
http://www.informationweek.de/index.php3?/channels/channel23/020418.htm

[ntwk02] Networkcomputing, Heft/02
http://www.networkcomputing.de/heft/solutions/sl-2002/sl_0502_52.htm

[SoftAG] Software AG Pressemitteilungen : *Anwenderforum; EAI-, XML- und Web-Services,*
http://www.softwareag.com/germany/news/pr_releases/08_02/EAIForum.htm

[SwIntbk] Swiss Interbank Clearing AG : Starke Zusammenarbeit-*Implementierung einer Web Services-Infrastruktur* , XML Magazin & Webservices, Ausgabe 2/02

[Wong01] Sam Wong, Patkai Networks: *The Next Evolution of Application Integration* (08/01)
http://e-serv.ebizq.net/wbs/wong_1.html

[WSw3c] www-Consortium (w3c): *Webservice Definition*
http://www.w3.org/TR/wsa-reqs#IDAIO2IB

[xmlw3c] www-Consortium (w3c): *XML*
http://www.w3.org/XML/

[zdnet] ZDNET: *Besser leben durch XML-Schemata*
http://www.zdnet.de/internet/artikel/java/200202/xmlschemata_01-wc.html

[Klute] Klute: *XSLT*
http://xml.klute-thiemann.de/w3c-de/REC-xslt-20020318/

[XSLT] w3c: *XSL-Transformation*
http://www.w3.org/TR/xslt

Ehrenwörtliche Erklärung

Hiermit erkläre ich, Servet Catal, geboren am 30. Juli 1968, ehrenwörtlich,

(1) dass ich meine Masterthesis mit dem Titel:

„Integration von Legacy Systemen in das Internet durch die Exposition von Transaktionen als Web Services.“

unter Anleitung von Prof. Dr. rer. pol. Paul Wenzel und Dipl. Phys. Robert Jürgens selbständig und ohne fremde Hilfe angefertigt habe und keine anderen als in der Abhandlung angeführten Hilfen benutzt habe;

(2) dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Konstanz, den 18. Oktober 2002